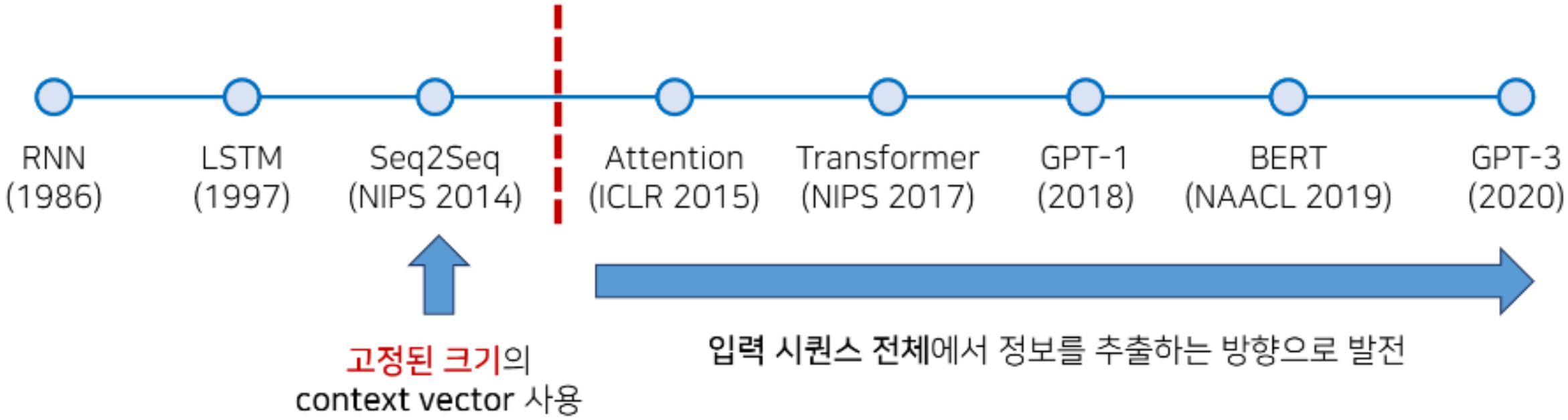
GDSC AI Week

고려대학교 정보대학
Korea University
College of Informatics

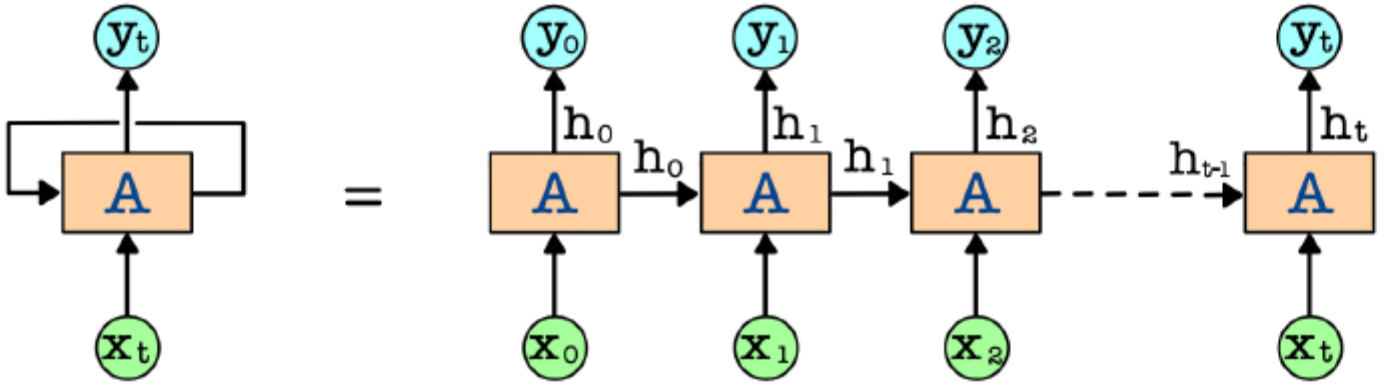
전병우, 이정재
ipcs@korea.ac.kr

딥러닝 기반의 기계 번역 발전 과정



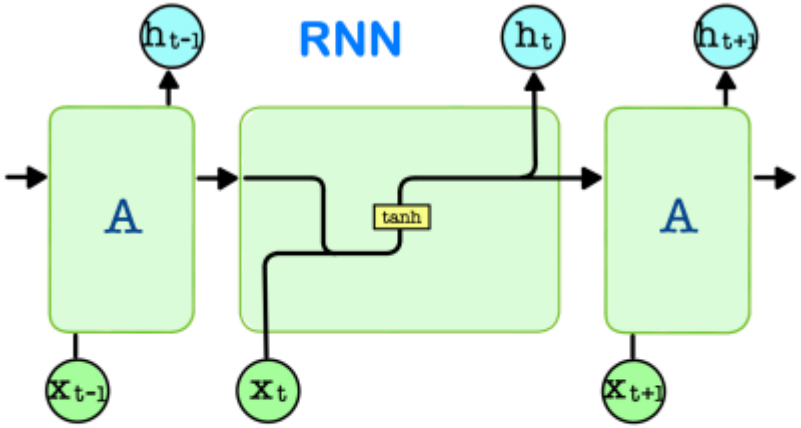
Recurrent Neural Network (Simple)

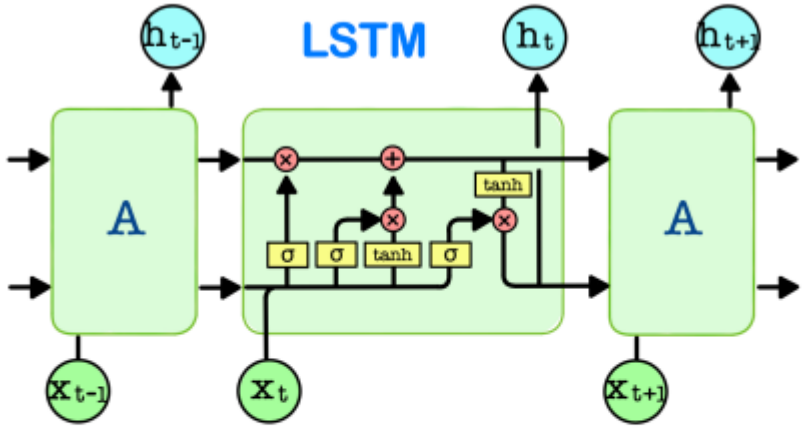
- RNN은 현재 입력과 이전 입력에 기반하여 예측을 수행하며, 순차적인 데이터에서 발생하는 시간적인 의존성을 학습.
- RNN은 현재 입력을 처리하면서 이전 정보를 메모리에 유지하고, 가변 길이의 시퀀스를 다룬다.
- RNN은 입력 시퀀스를 현재 입력인 x(현재 입력)를 기준으로 한 번에 하나의 데이터씩 처리하며, 지난 입력에 대한 메모리 역할을 하는 hidden state vector h를 유지한다.



Long Short-Term Memory (LSTM)

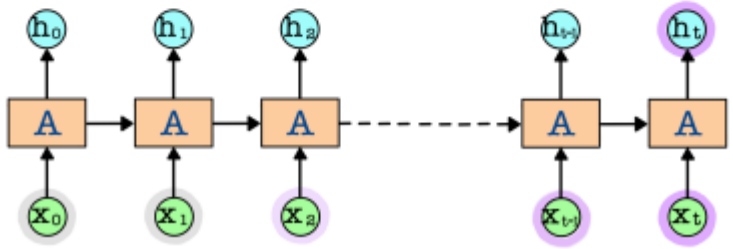
- LSTM은 hidden state vector에 더해 장기적인 정보를 기억하기 위한 cell state vector를 사용한다.
- Cell state vector는 long term dependency, vanishing, exploding gradient를 해결하는 데 사용된다.





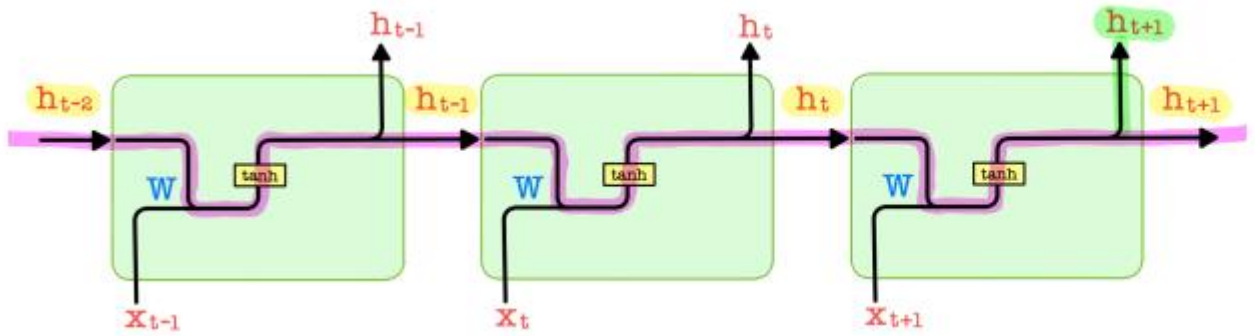
* Long-term dependency

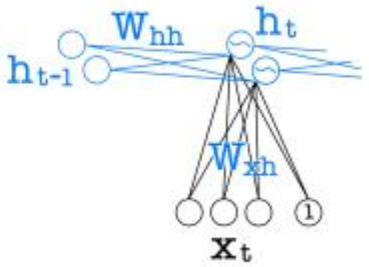
If the gap between the relevant information and the point where it is needed becomes large, RNN learning ability will be decreased.

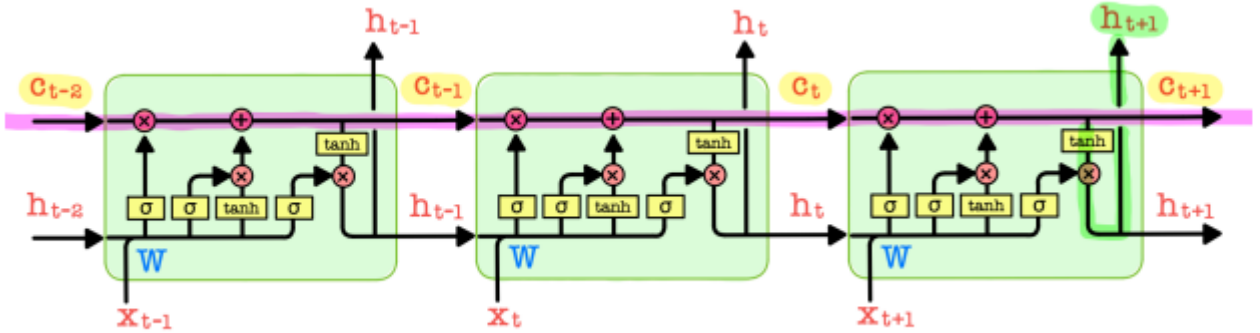


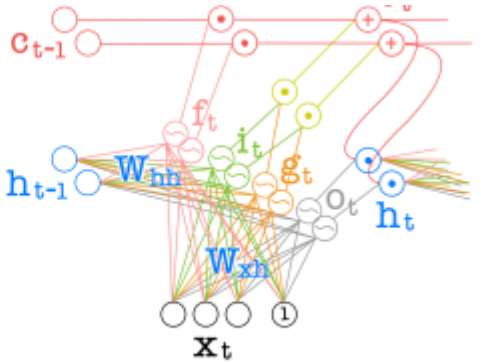
Long Short-Term Memory (LSTM)

- LSTM은 hidden state vector에 더해 장기적인 정보를 기억하기 위한 cell state vector를 사용한다.
- Cell state vector는 long term dependency, vanishing, exploding gradient를 해결하는 데 사용된다.



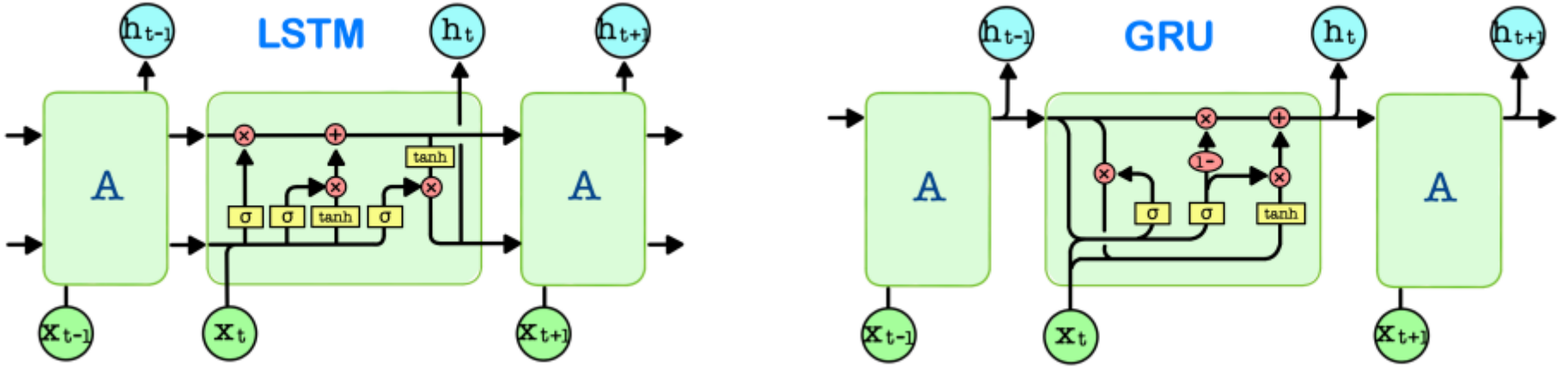






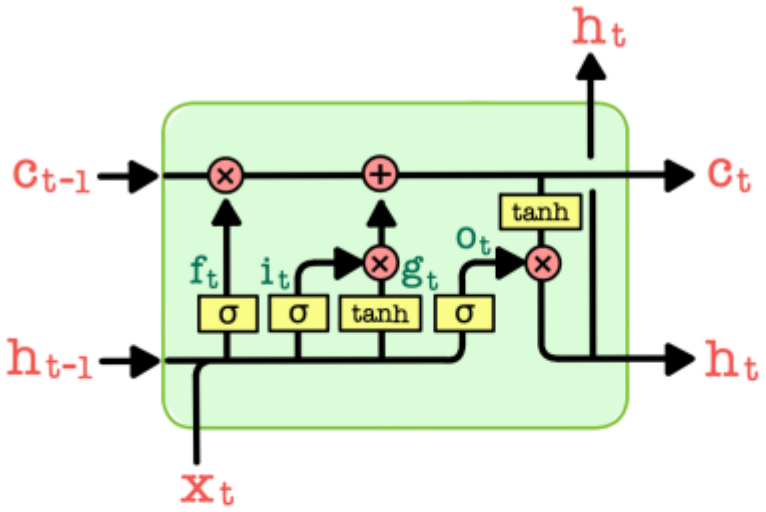
Gated Recurrent Unit (GRU)

- GRU(Gated Recurrent Unit)는 LSTM의 간소화된 변형 중 하나다.
- Cell state와 hidden state를 합치고 input 게이트와 forget 게이트를 단일한 update 게이트로 통합

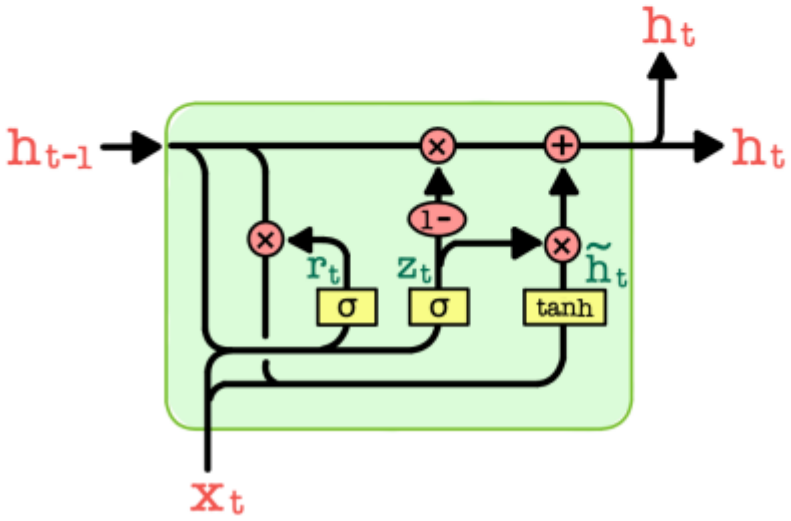


Gated Recurrent Unit (GRU)

딥러닝 수학
DL math
Korea University
Seungsang Oh



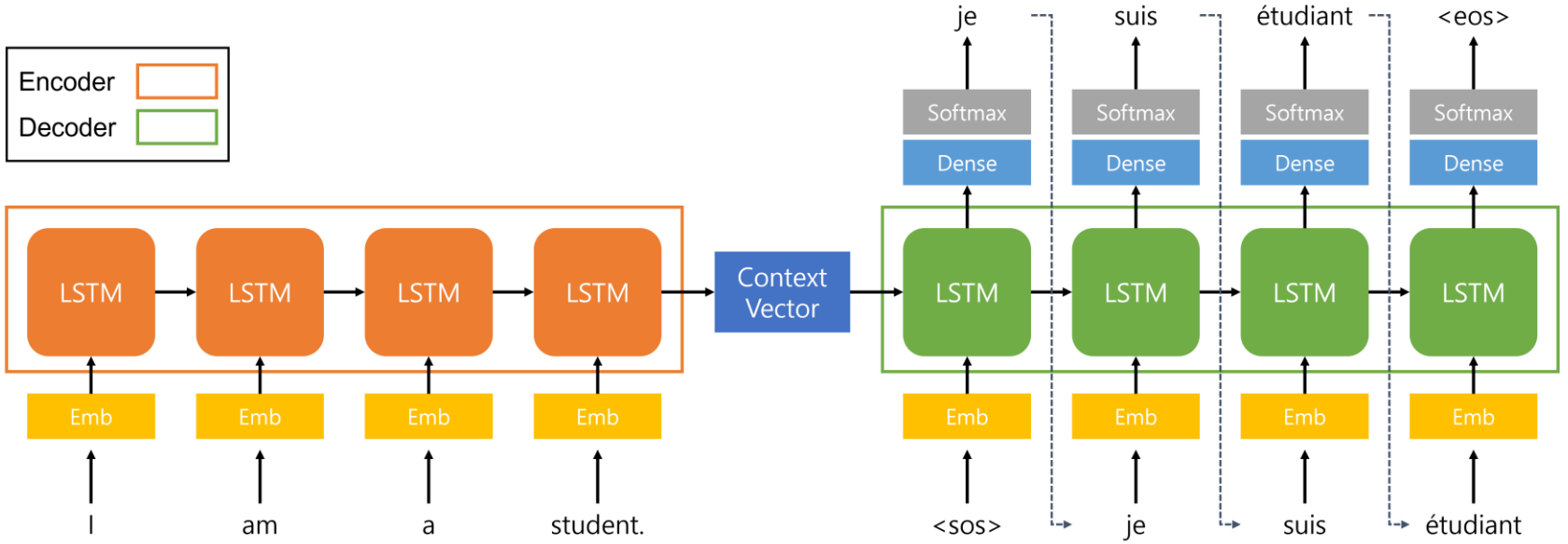
$$
\begin{aligned}
i_t &= \sigma(W_{xh}^i x_t + W_{hh}^i h_{t-1} + b_h^i) && \text{input gate} \\
f_t &= \sigma(W_{xh}^f x_t + W_{hh}^f h_{t-1} + b_h^f) && \text{forget gate} \\
o_t &= \sigma(W_{xh}^o x_t + W_{hh}^o h_{t-1} + b_h^o) && \text{output gate} \\
g_t &= \tanh(W_{xh}^g x_t + W_{hh}^g h_{t-1} + b_h^g) && \Longleftarrow \text{RNN core} \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t && \text{cell state} \\
h_t &= o_t \odot \tanh(c_t) && \text{hidden state}
\end{aligned}
$$



$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) && \text{update gate} \leftarrow \text{input gate, forget gate} \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) && \text{reset gate} \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) && \Longleftarrow \text{RNN core} \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{hidden state} \leftarrow \text{cell state, hidden state}
\end{aligned}
$$

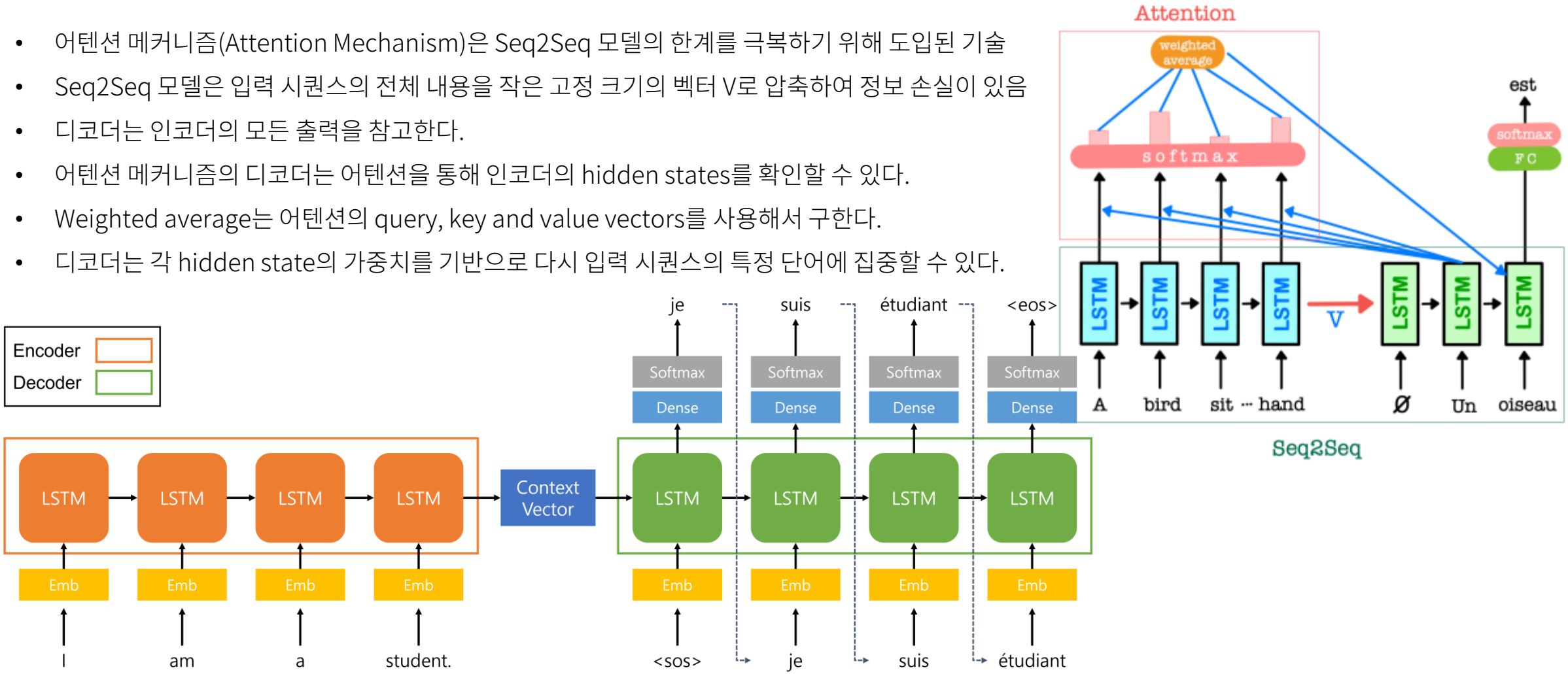
Seq2Seq for Machine Translation

- Encoder LSTM은 입력 문장을 단어 단위로 처리하고 입력 시퀀스의 전체 내용을 고정된 크기의 벡터(encoded 벡터)로 변환(word imbedding)
- Decoder LSTM은 이 Encoded 벡터를 기반으로 출력 단어를 예측하며, 생성한 컨텍스트 벡터를 받아 출력 시퀀스(번역문)를 출력한다.
- Encoder 네트워크는 영어를 읽고, Decoder 네트워크는 프랑스어를 작성한다.



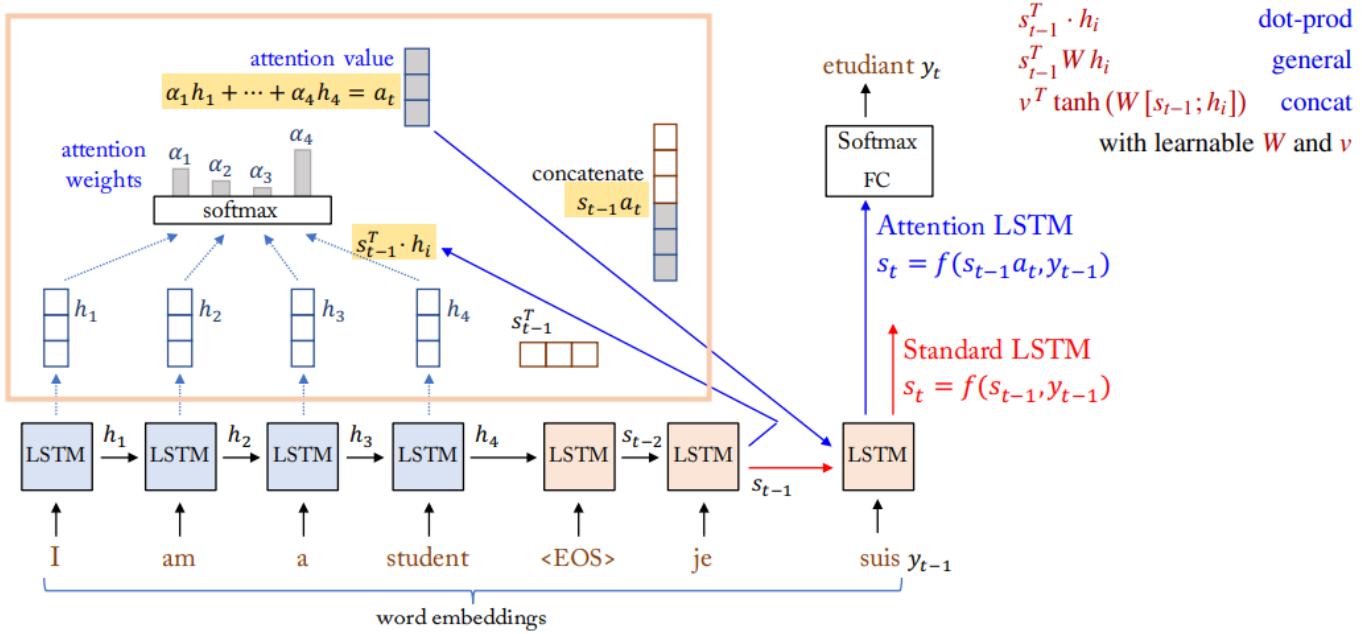
Seq2Seq with Attention Mechanism

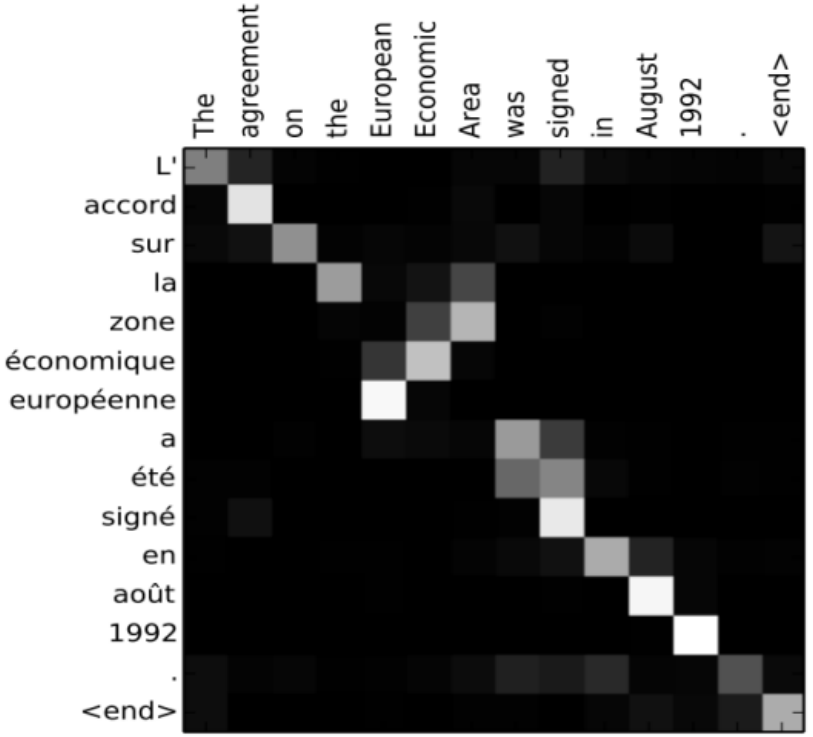
- 어텐션 메커니즘(Attention Mechanism)은 Seq2Seq 모델의 한계를 극복하기 위해 도입된 기술
- Seq2Seq 모델은 입력 시퀀스의 전체 내용을 작은 고정 크기의 벡터 V로 압축하여 정보 손실이 있음
- 디코더는 인코더의 모든 출력을 참고한다.
- 어텐션 메커니즘의 디코더는 어텐션을 통해 인코더의 hidden states를 확인할 수 있다.
- Weighted average는 어텐션의 query, key and value vectors를 사용해서 구한다.
- 디코더는 각 hidden state의 가중치를 기반으로 다시 입력 시퀀스의 특정 단어에 집중할 수 있다.



Attention Mechanism

- 어텐션 메커니즘을 사용하여 모델의 내부 동작을 볼 수 있다.
- 가중치에 기반한 어텐션은 특정 출력에 관련된 입력 부분이 어디인지 알 수 있다.
- 고정된 크기의 컨텍스트 벡터에 입력 시퀀스의 모든 정보를 압축하는 부담을 줄여준다.
- 디코더는 인코더의 모든 시점의 Hidden State를 활용하여 다음 단어를 예측한다.



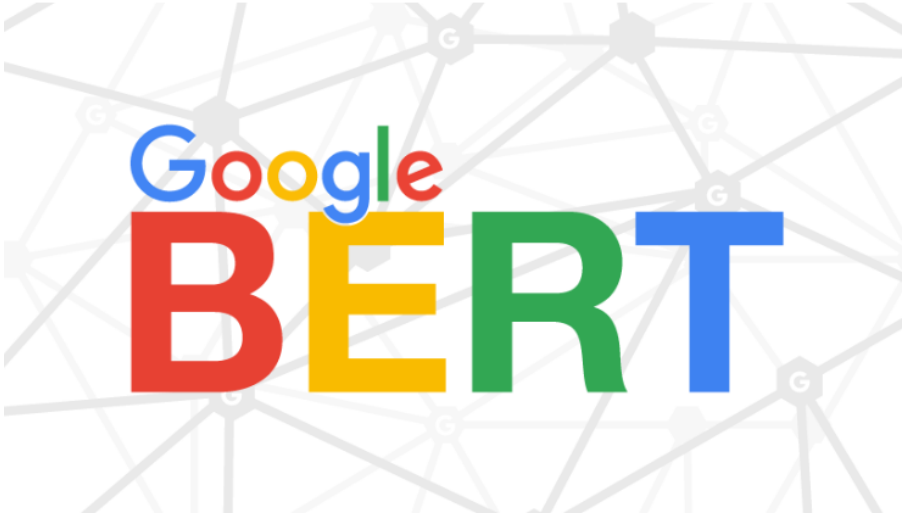


142 / 279

Attention Mechanism, Transformer

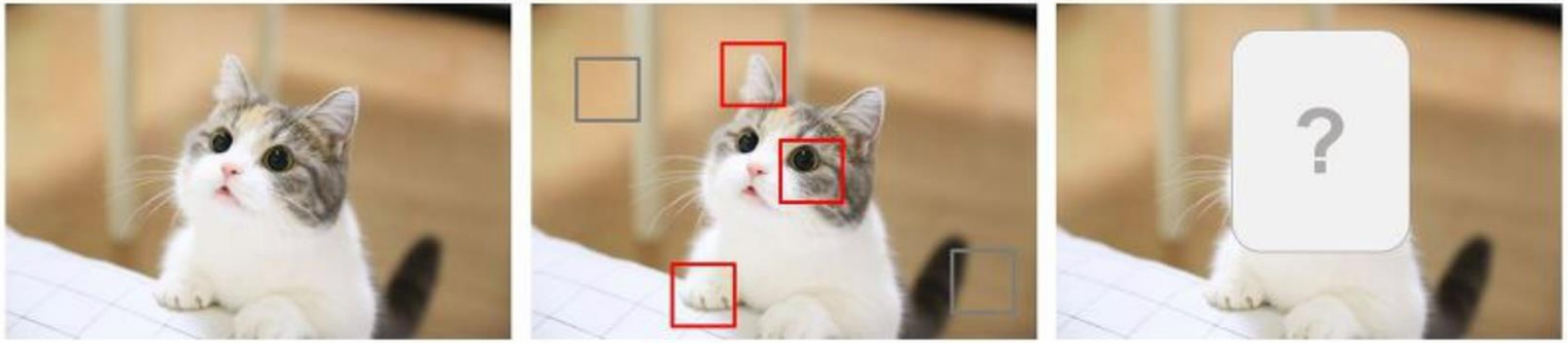
- 어텐션에서는 기존의 LSTM이 전체 문장을 하나의 은닉 상태로 인코딩하는 대신 각 입력 단어의 은닉 상태를 디코더의 각 단계에서 재사용
- 어텐션은 입력 단어를 처리하는 데에 병렬 컴퓨팅을 사용할 수 없어 대용량 코퍼스(많은 양의 텍스트 데이터)에 대한 훈련 시간이 증가
- 어텐션은 개별 내적을 통해 계산하므로 순서, 주변 단어가 반영되지 않음.
- 이러한 문제를 해결하기 위해 Transformer 구조가 제안되었고,
  - RNN 구조 없이 전적으로 셀프 어텐션(self-attention)을 기반으로 하여 병렬 컴퓨팅 및 전역 의존성 학습을 가능하게 한다.
  - 인코더와 디코더로 구성되었고, 여러 레이어에서 어텐션 과정을 반복
- BERT(Bidirectional Encoder Representations from Transformers)와
  GPT 모델은 이러한 Transformer의 아이디어를 사용하여 개발된 언어 이해 모델.

ChatGPT



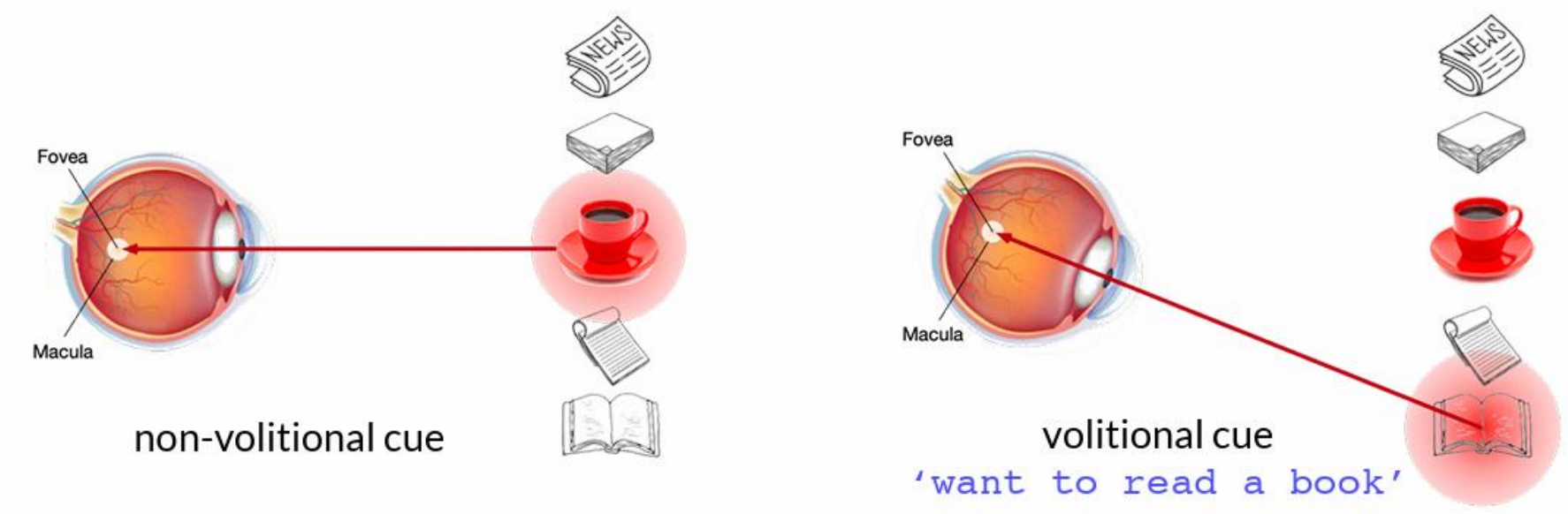
Attention in Cognitive Neuroscience

- 인지 과학 분야에서 과학자들은 attention에 대해 1890년부터 연구해왔다.

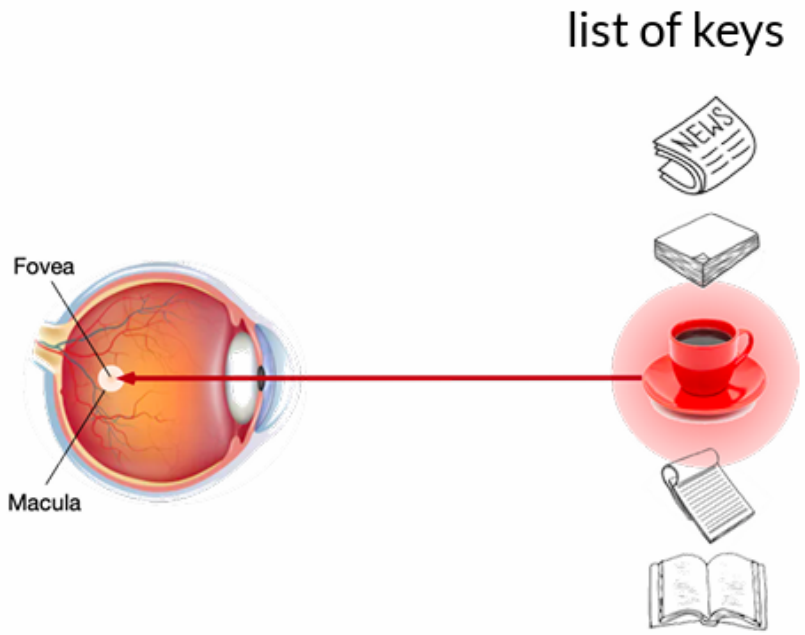


Attention in Cognitive Neuroscience

- 인지 과학 분야에서 과학자들은 attention에 대해 1890년부터 연구해왔다.
- Attention은 인간이 인식의 우선순위를 정하는 프로세스를 뜻한다.



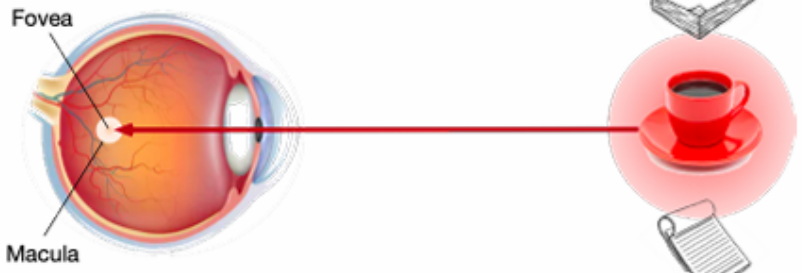
How can we formulate Attention?



$$\mathbf{k}_i$$

key index of key

How can we formulate Attention?



list of keys list of values

$g_\phi(\mathbf{k}_1)$

$g_\phi(\mathbf{k}_2)$

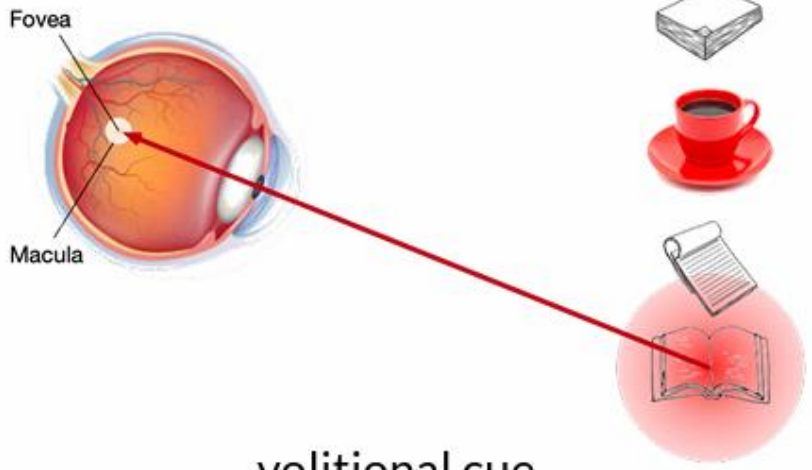
$\vdots$

$g_\phi(\mathbf{k}_i)$

i-th value $g_\phi(\mathbf{k}_i)$

value paramater key index of key

How can we formulate Attention?



list of keys list of values

$g_\phi(\mathbf{k}_1)$

$g_\phi(\mathbf{k}_2)$

$\vdots$

$g_\phi(\mathbf{k}_i)$

volitional cue

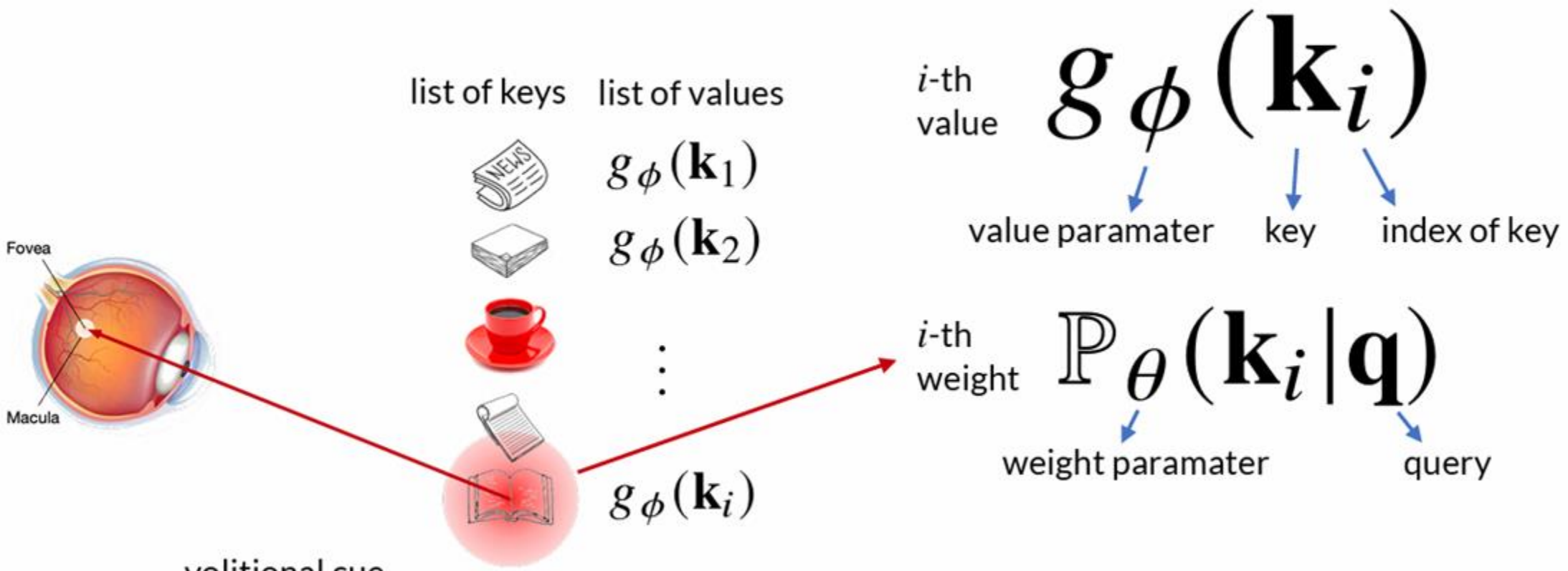
'want to read a book'

$\mathbf{q}$ query

i-th value $g_\phi(\mathbf{k}_i)$

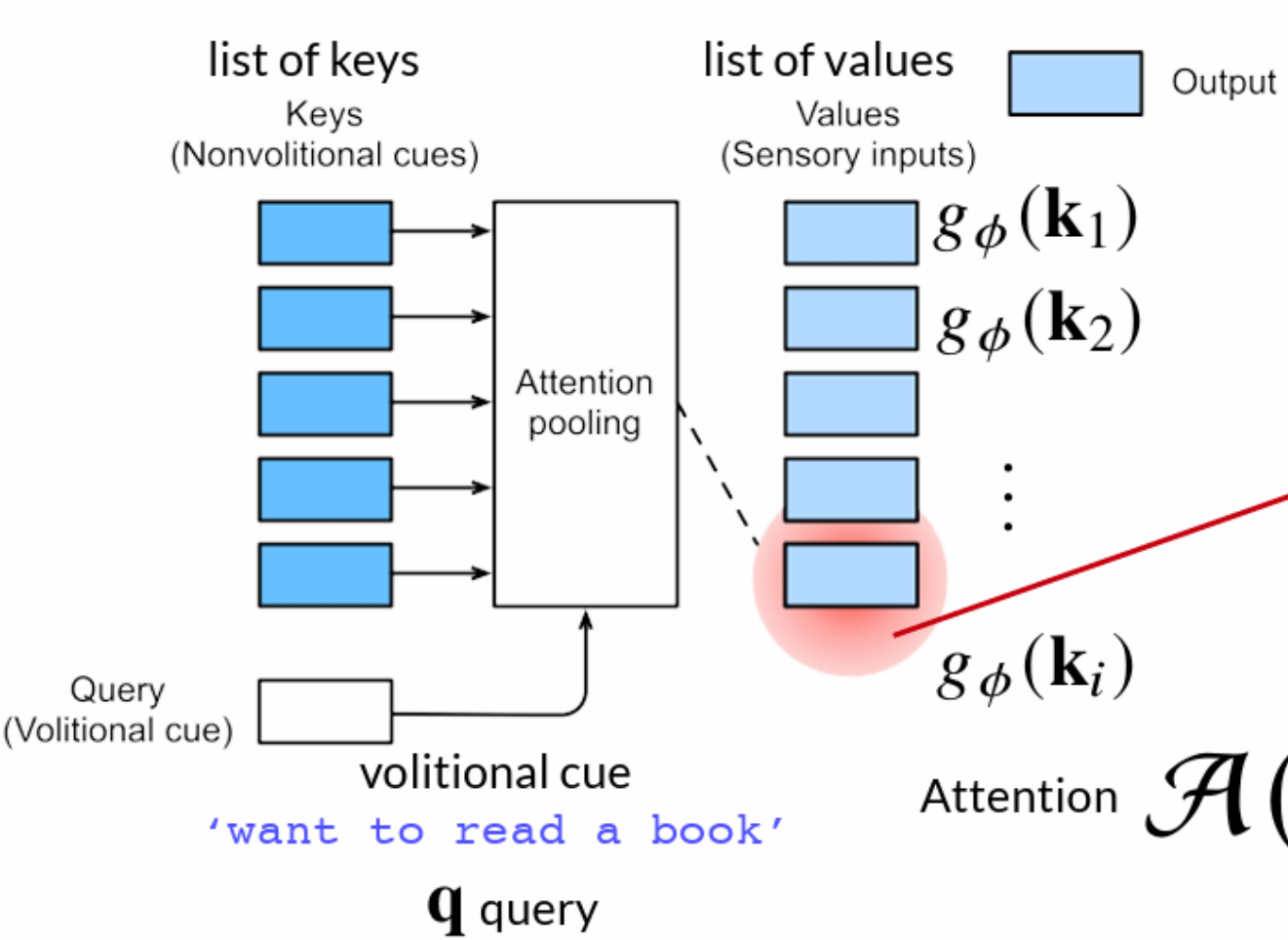
value paramater key index of key

How can we formulate Attention?



가중치는 조건부 확률로 계산할 수 있다!

How can we formulate Attention?



i-th value $g_\phi(\mathbf{k}_i)$

value paramater, key, index of key

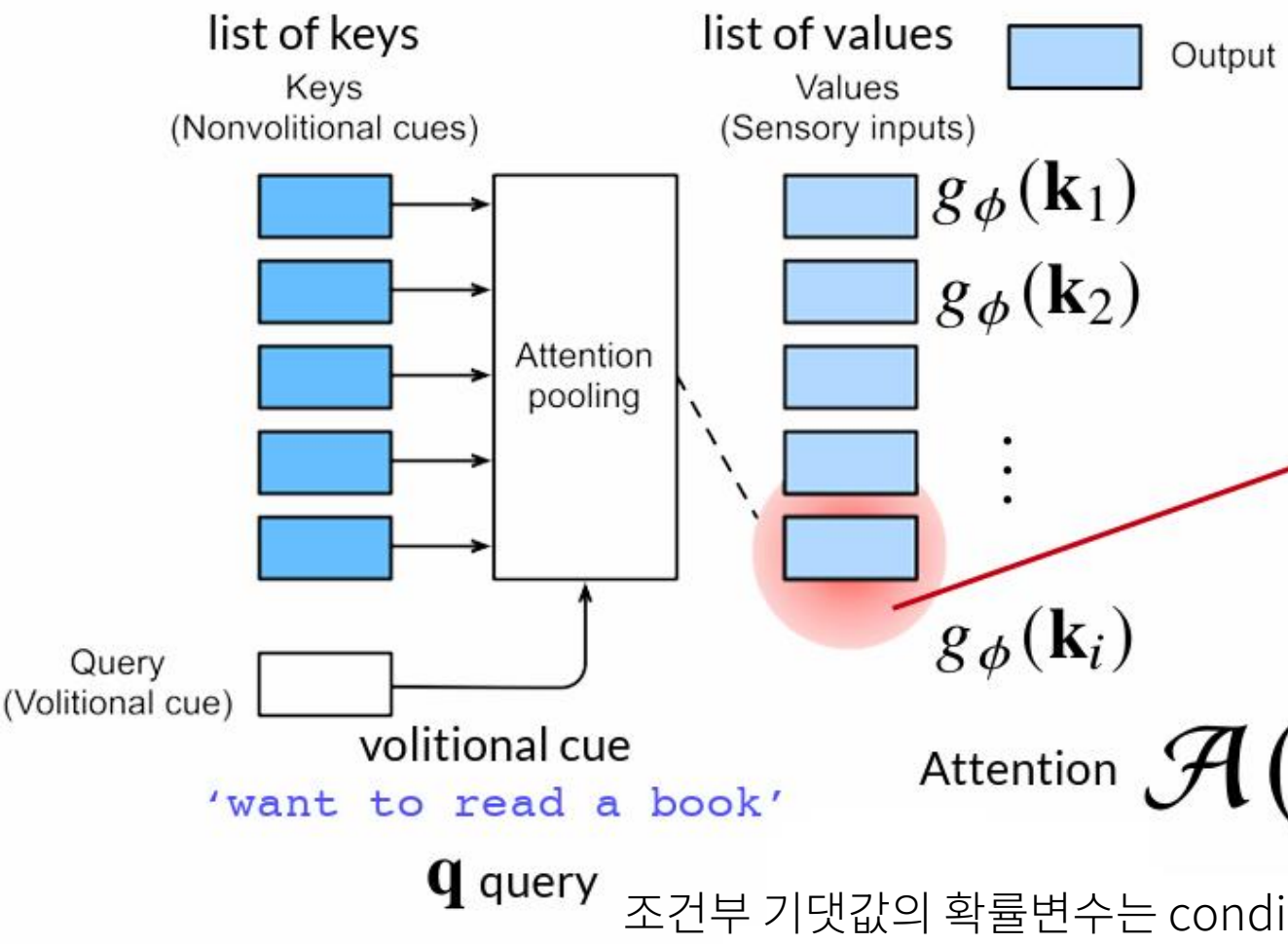
i-th weight $\mathbb{P}_\theta(\mathbf{k}_i|\mathbf{q})$

weight paramater, query

Attention $\mathcal{A}(\mathbf{q}) = \mathbb{E}_{\mathbf{k}\sim\mathbb{P}_\theta(\cdot|\mathbf{q})}[g_\phi(\mathbf{k})]$

weighted expectation = pooling

How can we formulate Attention?



i-th value $g_\phi(\mathbf{k}_i)$

value paramater, key, index of key

i-th weight $\mathbb{P}_\theta(\mathbf{k}_i|\mathbf{q})$

weight paramater, query

Attention $\mathcal{A}(\mathbf{q}) = \mathbb{E}_{\mathbf{k}\sim\mathbb{P}_\theta(\cdot|\mathbf{q})}[g_\phi(\mathbf{k})]$

조건부 기댓값의 확률변수는 conditioning variable이다!

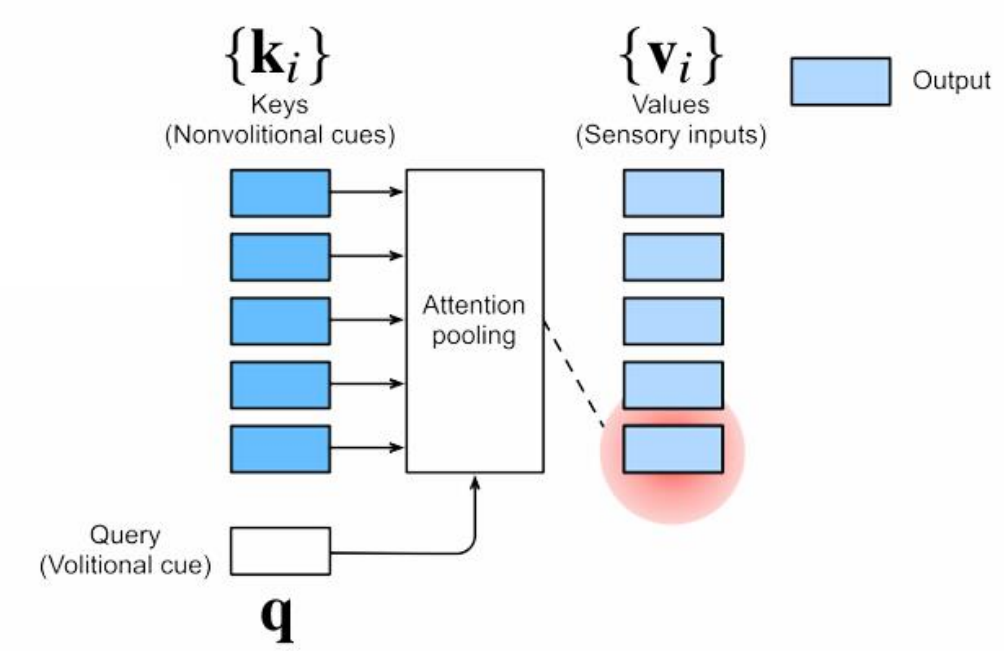
weighted expectation = pooling

How can we design Attention module?

- Attention은 input에 대해 bias alignment로 이루어진 pooling으로 만들 수 있다.
  - Keys: list of cues $\{\mathbf{k}_i\} \subset \mathbb{R}^{\mathbf{d}_k}$
  - Query: volitional cues $\mathbf{q} \in \mathbb{R}^{\mathbf{d}_q}$
  - Values: feature representation $\{\mathbf{v}_i\} \subset \mathbb{R}^{\mathbf{d}_v}$

이 때, key와 values의 개수는 같아야 한다.

$$\Rightarrow \#(\{\mathbf{k}_i\}) = \#(\{\mathbf{v}_i\})$$

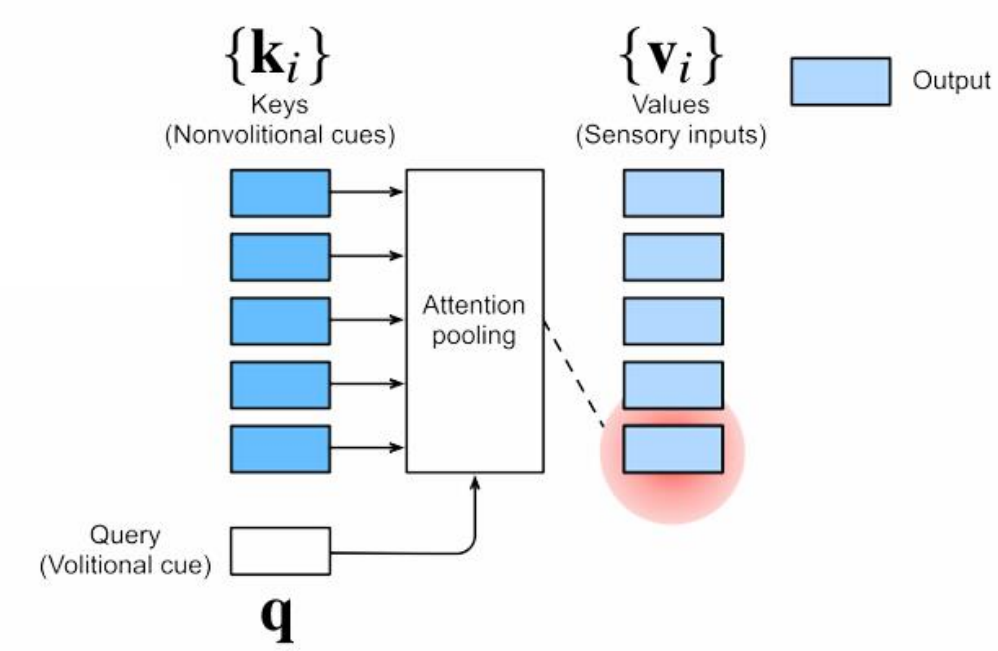


How can we design Attention module?

- Attention은 input에 대해 bias alignment로 이루어진 pooling으로 만들 수 있다.
  - Keys: list of cues $\{\mathbf{k}_i\} \subset \mathbb{R}^{\mathrm{d}_k}$
  - Query: volitional cues $\mathbf{q} \in \mathbb{R}^{\mathrm{d}_q}$
  - Values: feature representation $\{\mathbf{v}_i\} \subset \mathbb{R}^{\mathrm{d}_v}$

- (해석) 주어진 어떤 query에 대해 attention module은 attention pooling을 통해 represented features에 대한 bias selection을 수행한다.

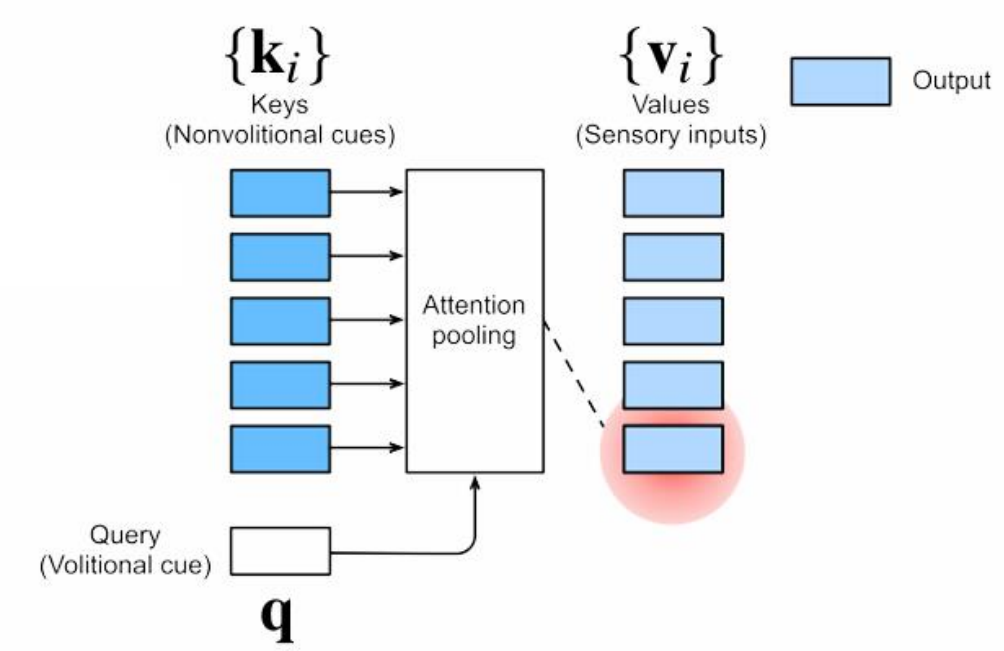
$$\mathcal{A}(\mathbf{q}) = \mathbb{E}_{\mathbf{k} \sim \mathbb{P}_\theta(\cdot|\mathbf{q})}[g_\phi(\mathbf{k})]$$



How can we design Attention module?

- Attention은 input에 대해 bias alignment로 이루어진 pooling으로 만들 수 있다.
  - Keys: list of cues $\{\mathbf{k}_i\} \subset \mathbb{R}^{\mathrm{d}_k}$
  - Query: volitional cues $\mathbf{q} \in \mathbb{R}^{\mathrm{d}_q}$
  - Values: feature representation $\{\mathbf{v}_i\} \subset \mathbb{R}^{\mathrm{d}_v}$

- (해석) 주어진 어떤 query에 대해 attention module은 attention pooling을 통해 represented features에 대한 bias selection을 수행한다.



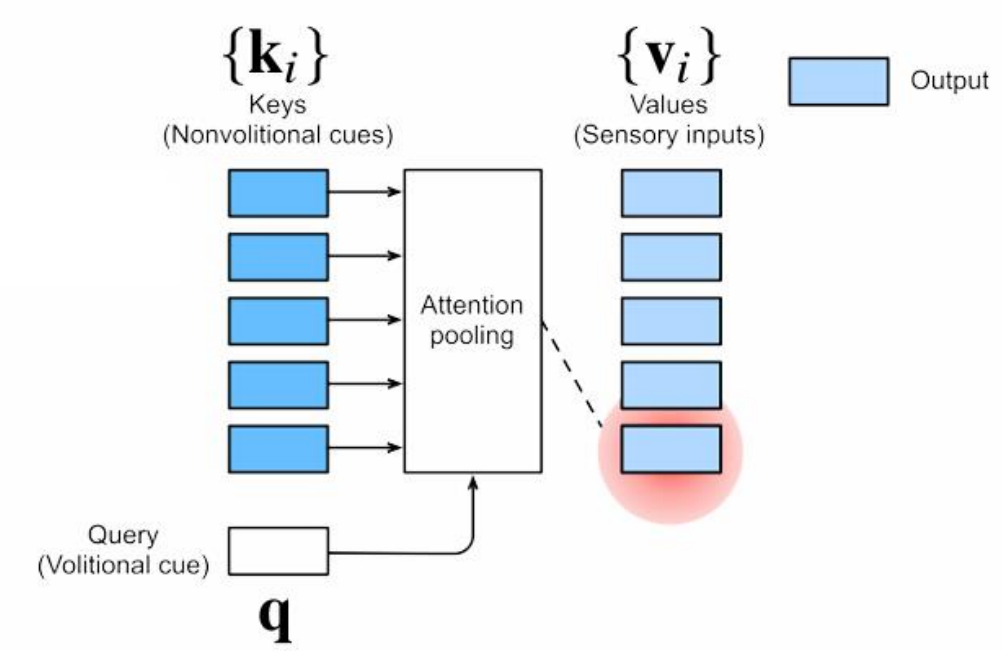
$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i$$

attention weights

How can we design Attention module?

- Attention은 input에 대해 bias alignment로 이루어진 pooling으로 만들 수 있다.
  - Keys: list of cues $\{\mathbf{k}_i\} \subset \mathbb{R}^{d_k}$
  - Query: volitional cues $\mathbf{q} \in \mathbb{R}^{d_q}$
  - Values: feature representation $\{\mathbf{v}_i\} \subset \mathbb{R}^{d_v}$

- (해석) 주어진 어떤 query에 대해 attention module은 attention pooling을 통해 represented features에 대한 bias selection을 수행한다.



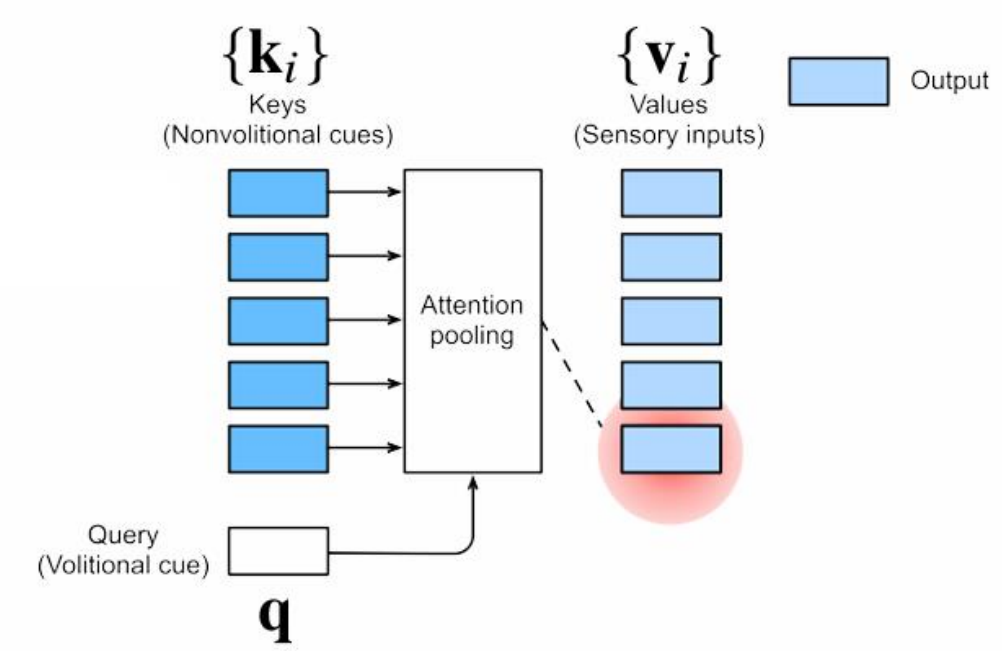
$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^m) = \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \qquad \alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i)) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^m \exp(a(\mathbf{q}, \mathbf{k}_j))}$$

attention weights　　attention scoring function

How can we design Attention module?

- Attention은 input에 대해 bias alignment로 이루어진 pooling으로 만들 수 있다.
  - Keys: list of cues $\{\mathbf{k}_i\} \subset \mathbb{R}^{d_k}$
  - Query: volitional cues $\mathbf{q} \in \mathbb{R}^{d_q}$
  - Values: feature representation $\{\mathbf{v}_i\} \subset \mathbb{R}^{d_v}$

- (해석) 주어진 어떤 query에 대해 attention module은 attention pooling을 통해 represented features에 대한 bias selection을 수행한다.



$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$

우리는 이것을 attention module이라 부른다

How to compute attention scoring $a(\mathrm{q}, \mathrm{k})$?

- Scaled Dot-product
  - $d_k = d_q = d$ 일 때 사용한다.

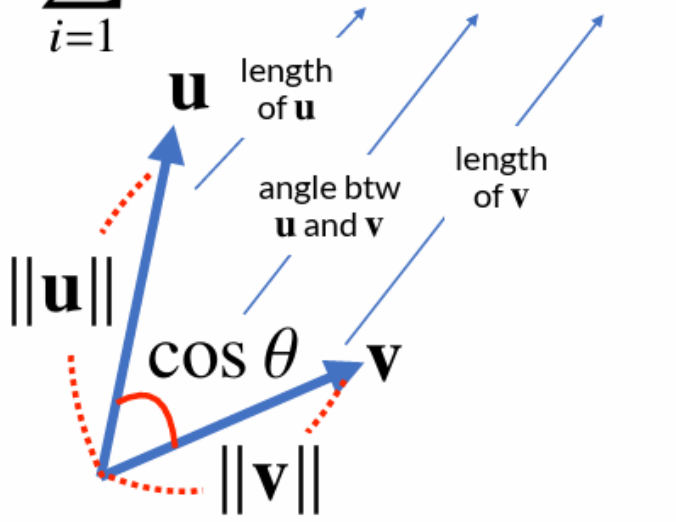
$$a(\mathbf{q}, \mathbf{k}) = \frac{\langle \mathbf{q}, \mathbf{k} \rangle}{\sqrt{d}}$$

- Additive Pooling
  - $d_k \neq d_q$ 일 때 사용한다.

$$a(\mathbf{q}, \mathbf{k}) = \langle \mathbf{w}, \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \rangle \quad (\mathbf{w} \in \mathbb{R}^{d_h},\ \mathbf{W}_q \in \mathbb{R}^{d_h \times d_q},\ \mathbf{W}_k \in \mathbb{R}^{d_h \times d_k})$$
$$= \mathbf{w}^\top \tanh(\mathbf{W}_q \mathbf{q} + \mathbf{W}_k \mathbf{k}) \in \mathbb{R}$$

$< \cdot, \cdot >$는 내적(dot-product, inner product)이다

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{d} u_i v_i = \|\mathbf{u}\| \cos\theta \|\mathbf{v}\|$$



How can we interpret attention modules?

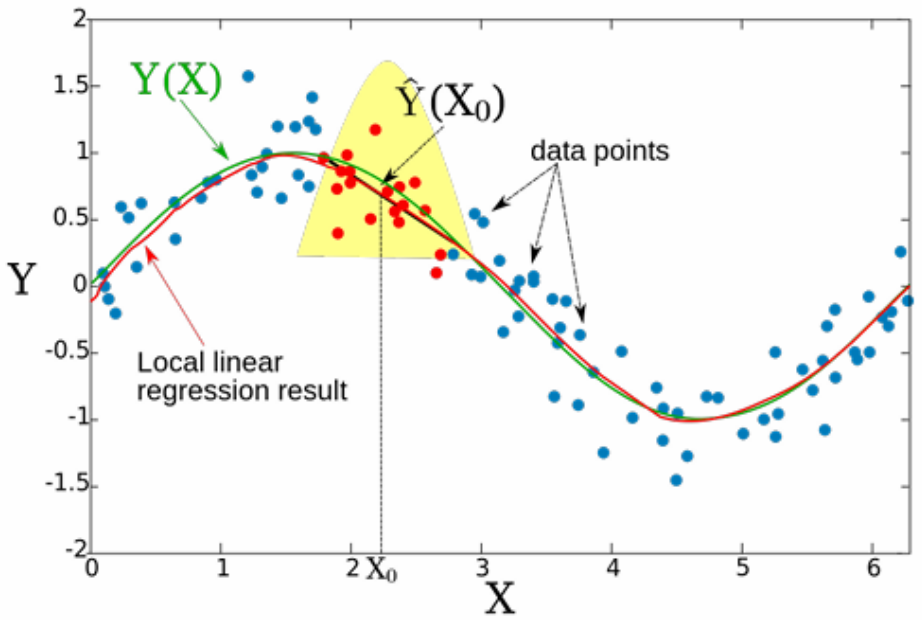
Attention이 등장하기 한참 이전에 이미 통계학에서는 유사한 회귀 방법이 존재했다.

- Attention Module

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$

- Nadaraya-Watson Kernel Regression (1964)

$$\hat{Y}(X) = \sum_{i=1}^{m} \frac{K(X, X_i)}{\sum_{j=1}^{m} K(X, X_j)} Y(X_i)$$



Kernel smoothing via local regression

How can we interpret attention modules?

- Attention Module

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$

attention weight

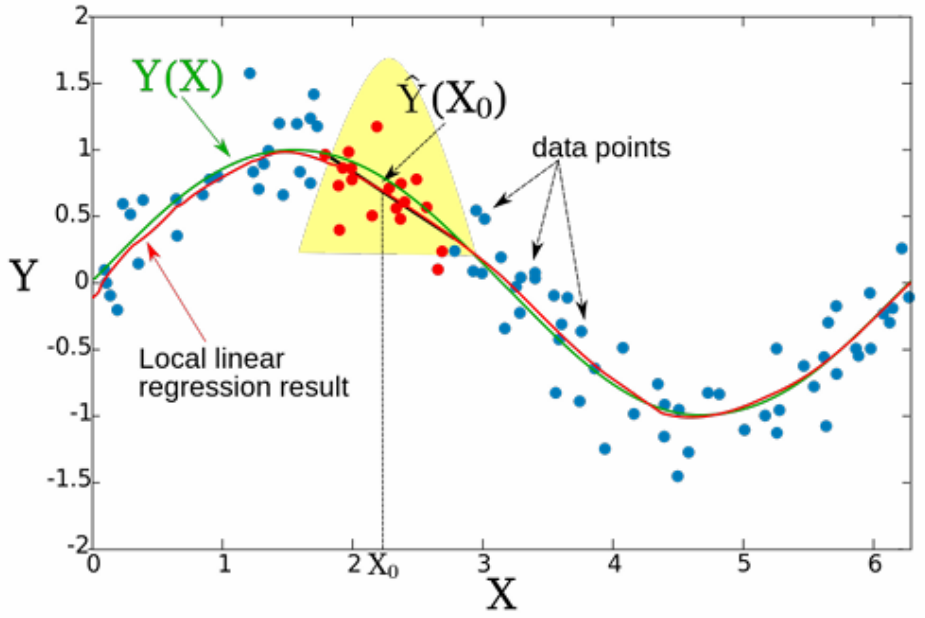
- Nadaraya-Watson Kernel Regression (1964)

$$\hat{Y}(X) = \sum_{i=1}^{m} \frac{K(X, X_i)}{\sum_{j=1}^{m} K(X, X_j)} Y(X_i)$$

kernel-weight

equivalent!

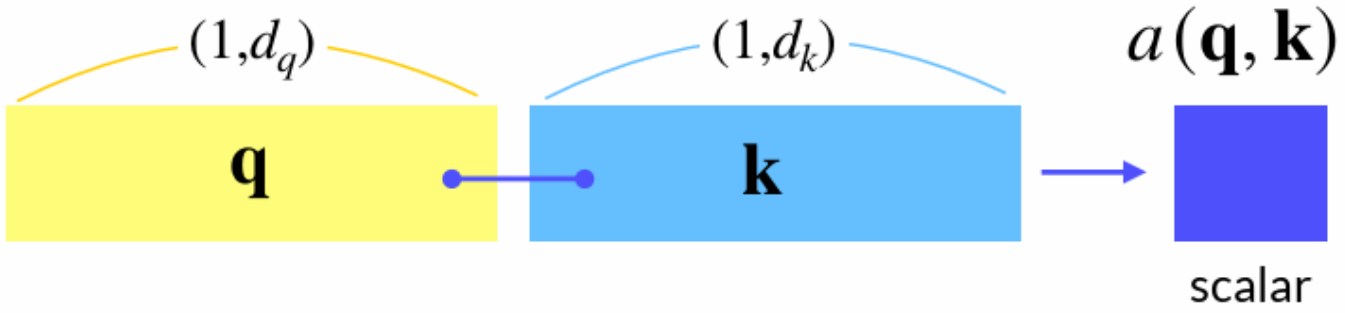
결국 attention은 softmax scoring을 이용한 learnable kernel smoother인 셈이다.



Kernel smoothing via local regression

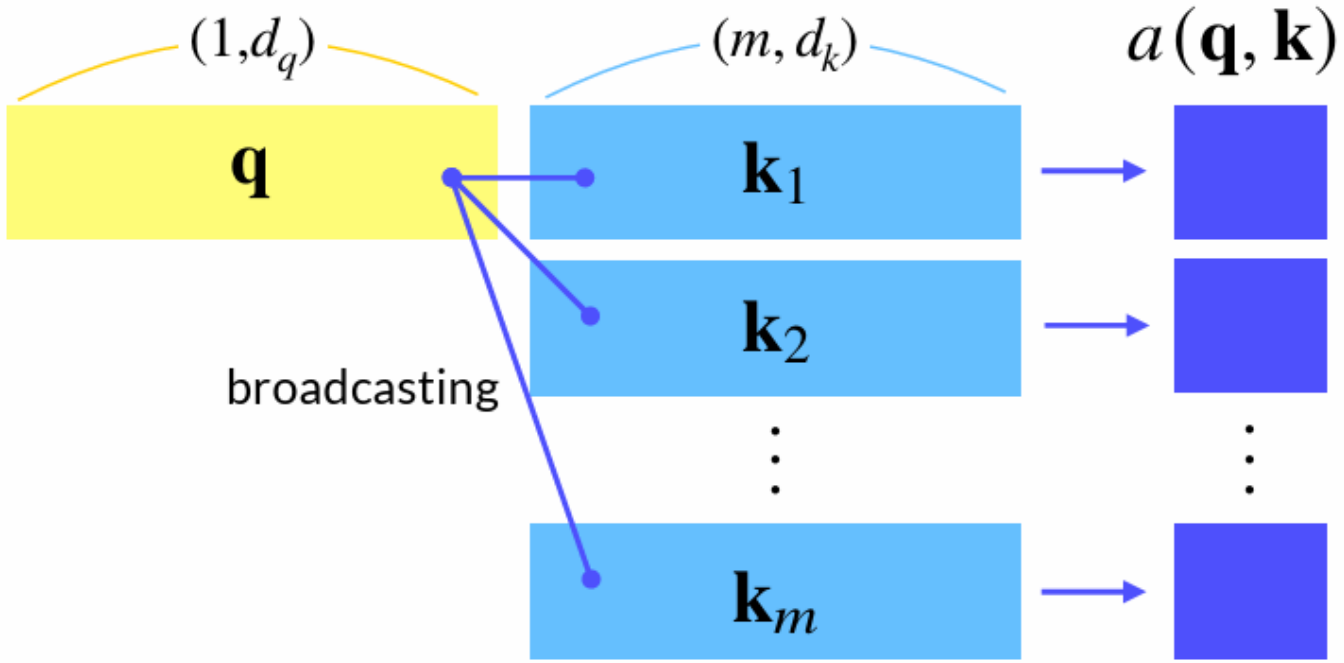
Let’s code attention modules!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



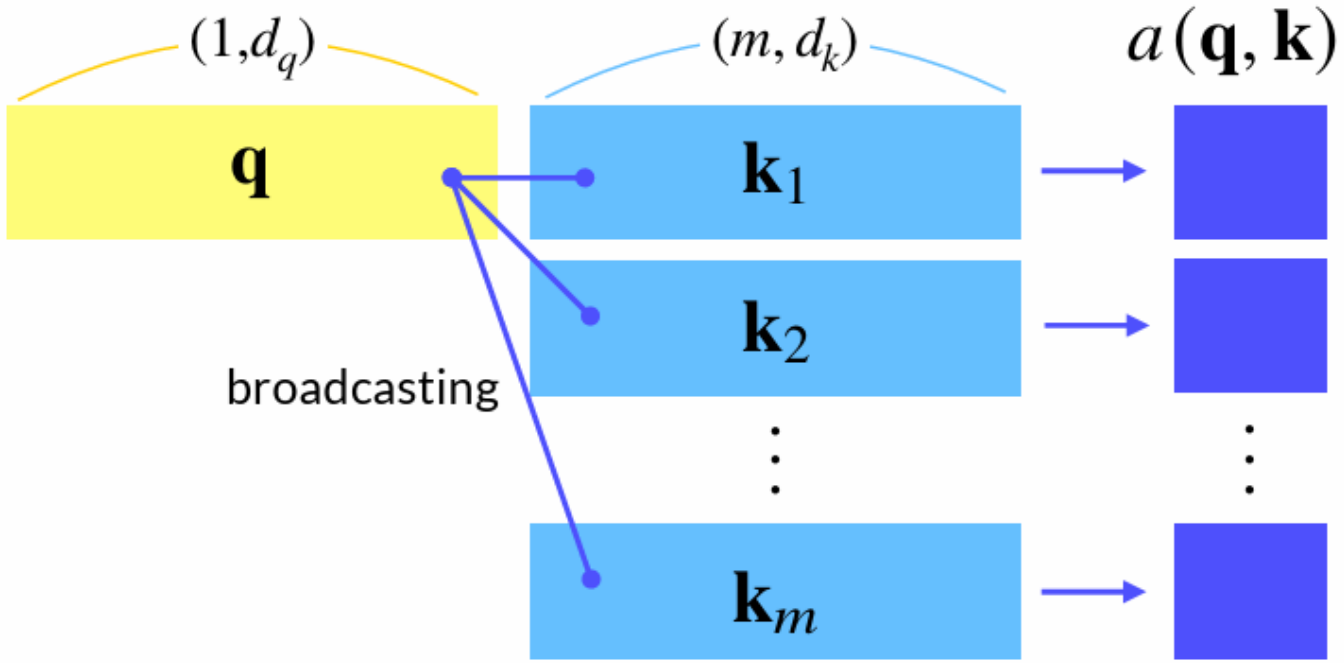
Let’s code attention modules!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



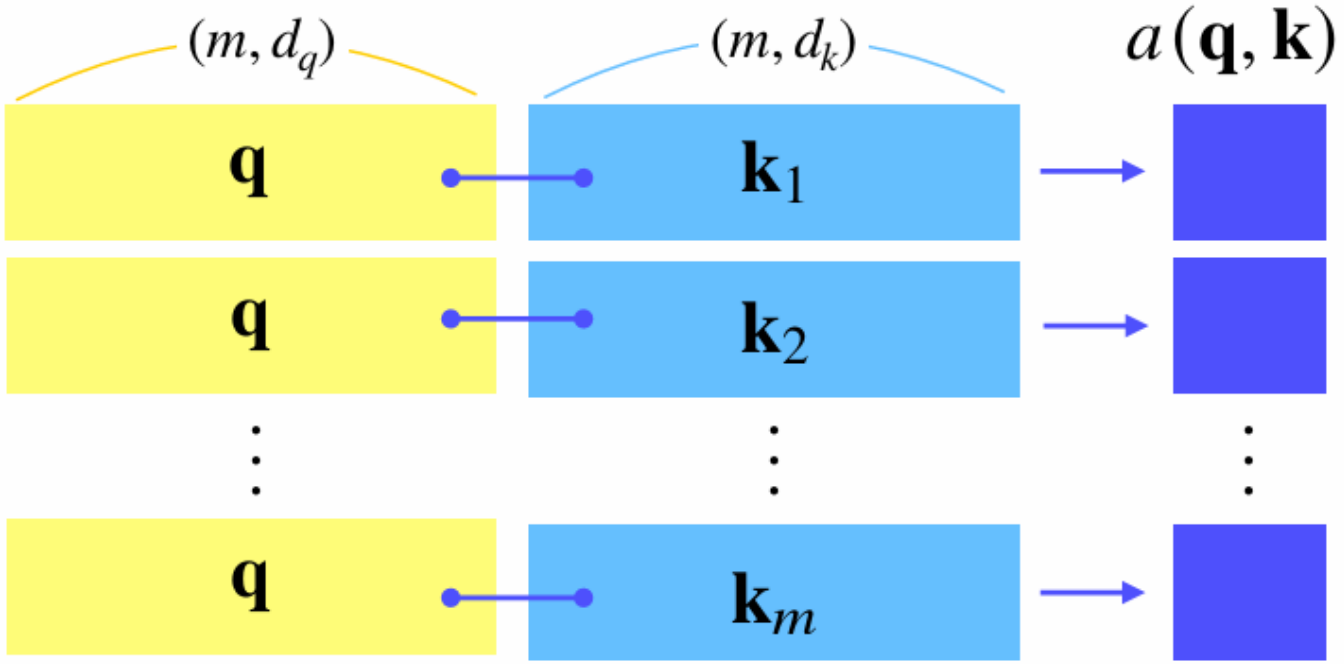
Let’s code attention modules!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



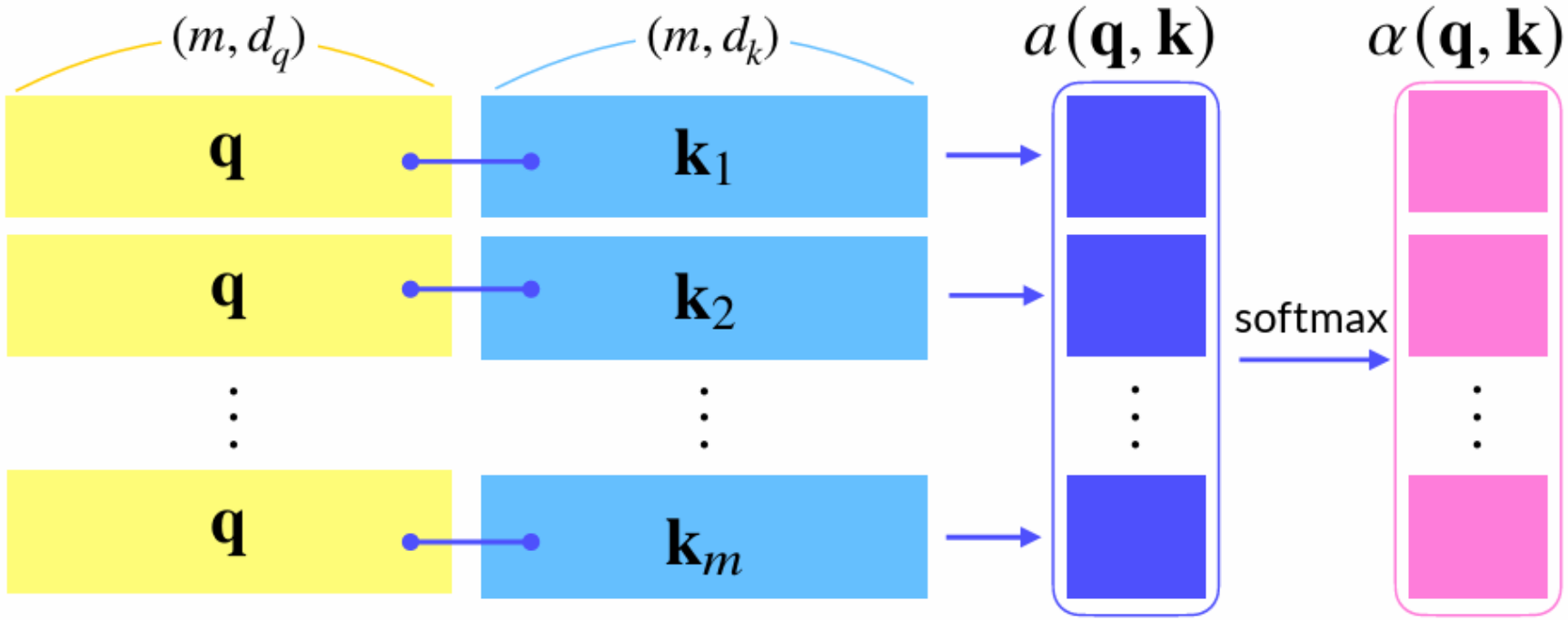
Let’s code attention modules!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



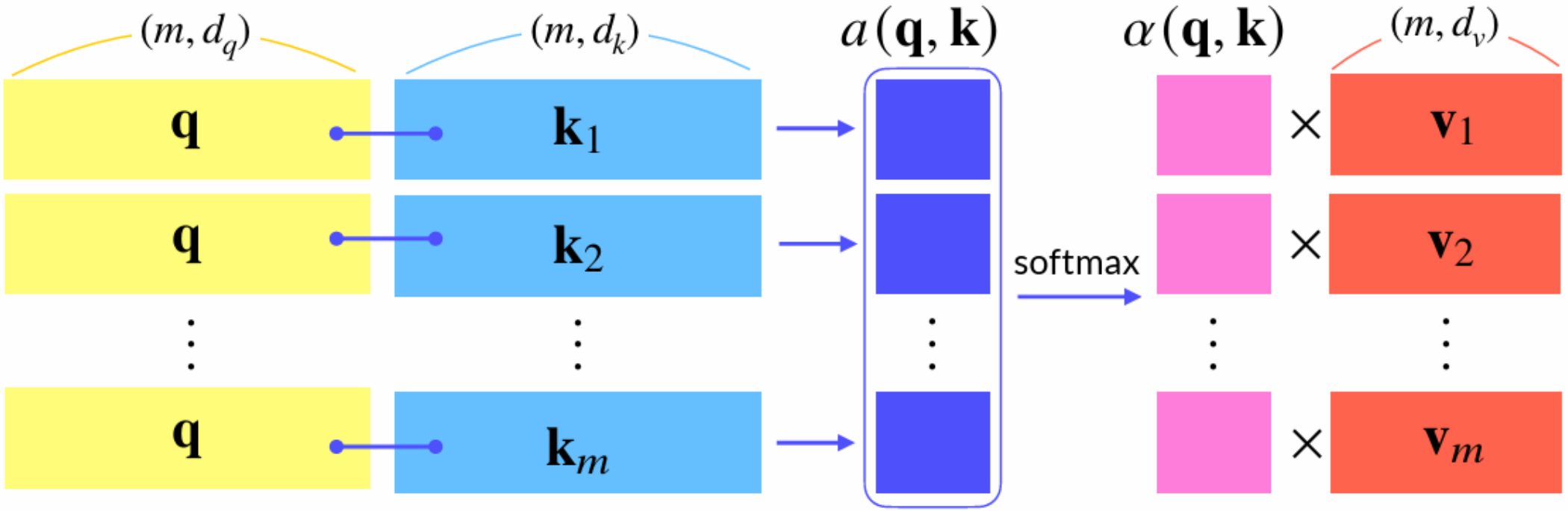
Let’s code attention modules!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



Let’s code attention modules!

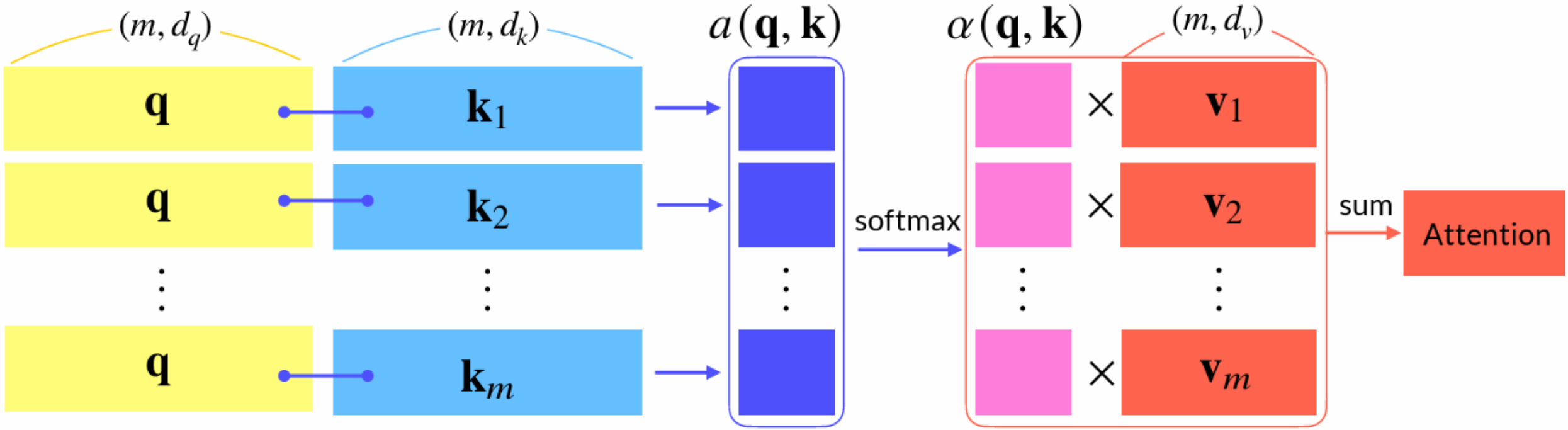
$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



Let’s code attention modules!

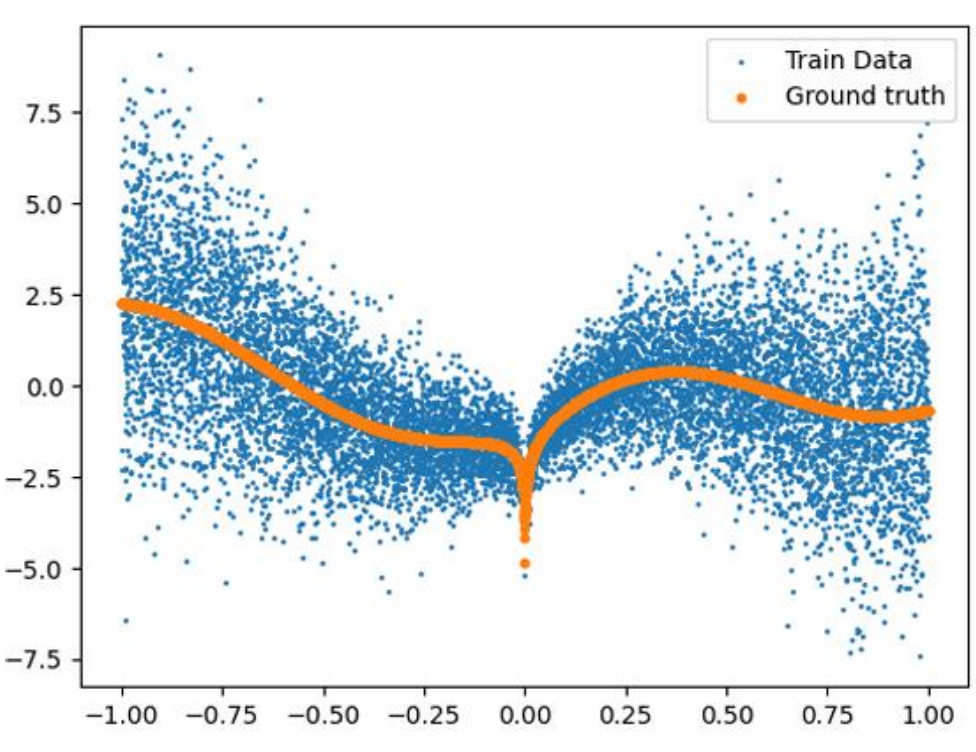
지금까지의 설명은 batch size가 1일 때임을 유의하자!

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$



Let’s code attention modules!

Ground Truth $y = \frac{4x^2}{5} - \frac{x}{2} + \sin(5x) + \frac{1}{2}\log|x| + \epsilon(x) \quad \epsilon(x) \sim (0.5 + 2|x|)\mathcal{N}(0,1)$



이 함수는

1. 0에서 불연속
2. Nonlinear 모양
3. 이분산성(heteroscedastic) noise

때문에 학습하기 어렵다

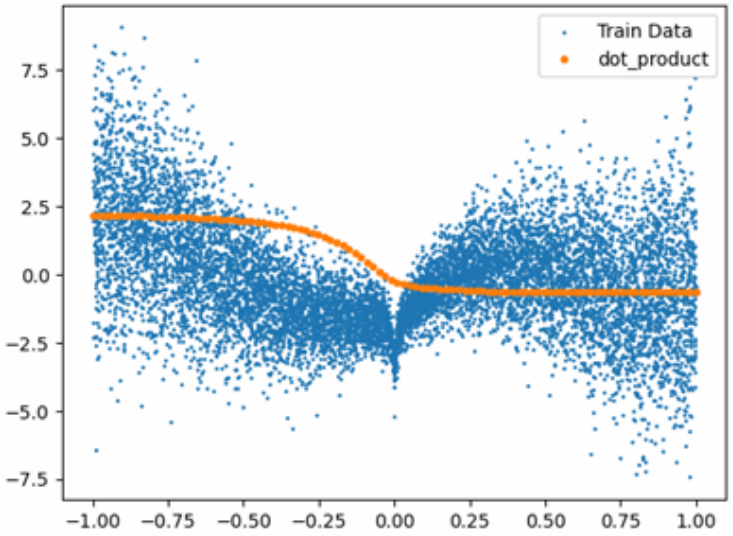
Let’s code attention modules!

우리는 파라미터를 사용하지 않았음에 유의하자!

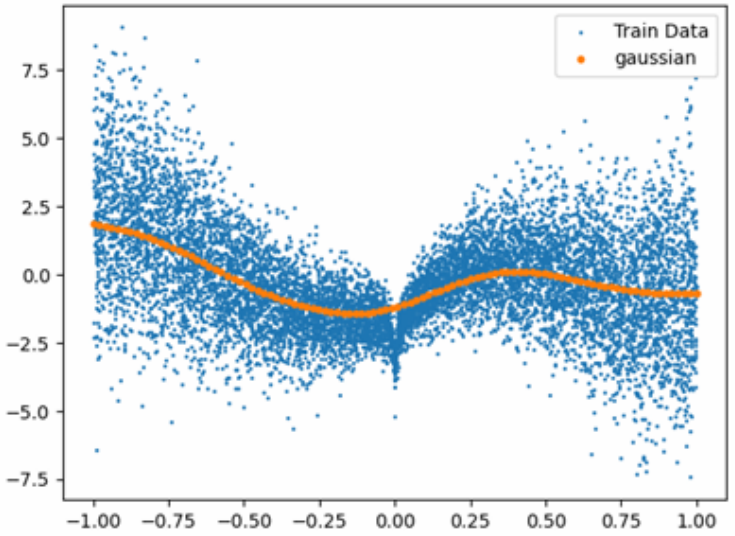
$$f(\mathbf{x}, \{(\mathbf{x}_i, y_i)_{i=1}^{m}\}) = \sum_{i=1}^{m} \mathrm{softmax}(a(\mathbf{x}, \mathbf{x}_i))y_i$$

# of data: m

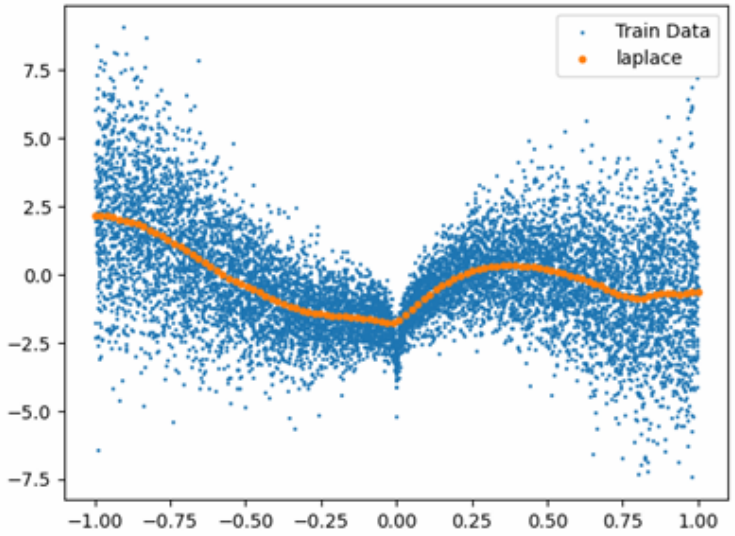
query: $\mathbf{x}$ key: $\mathbf{x}_i$ value: y_i attention scoring: $a(\mathbf{x}, \mathbf{x}_i)$



$$a(\mathbf{q}, \mathbf{k}) = \langle \mathbf{q}, \mathbf{k} \rangle$$



$$a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|_2^2$$



$$a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|$$

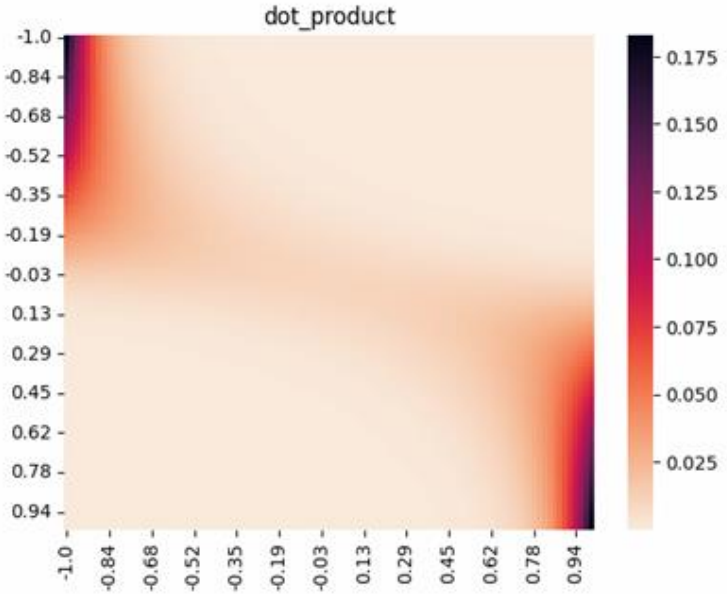
Let’s code attention modules!

우리는 파라미터를 사용하지 않았음에 유의하자!

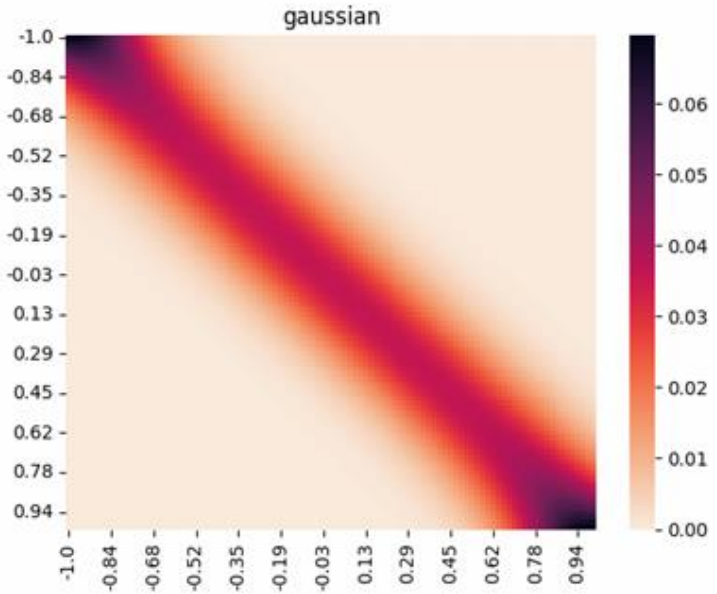
$$f(\mathbf{x}, \{(\mathbf{x}_i, y_i)_{i=1}^{m}\}) = \sum_{i=1}^{m} \mathrm{softmax}(a(\mathbf{x}, \mathbf{x}_i))\, y_i$$

# of data: m

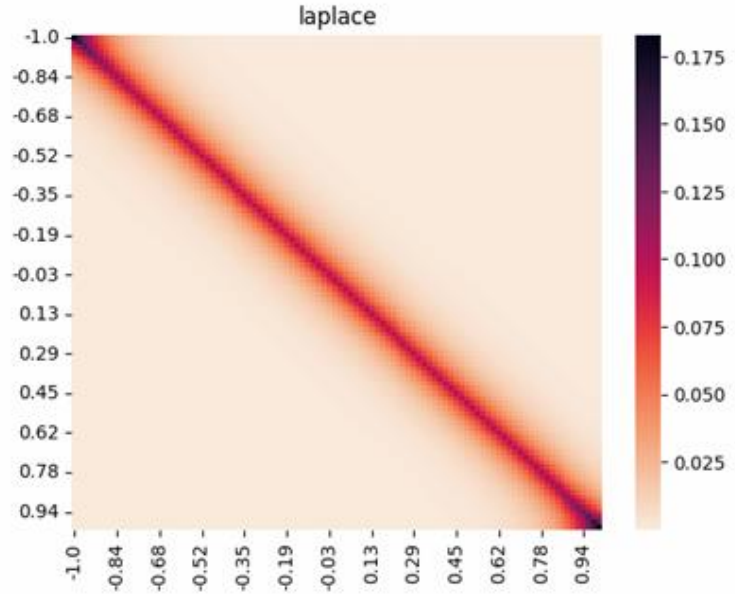
query: $\mathbf{x}$ key: $\mathbf{x}_i$ value: y_i attention scoring: $a(\mathbf{x}, \mathbf{x}_i)$



$$a(\mathbf{q}, \mathbf{k}) = \langle \mathbf{q}, \mathbf{k} \rangle$$



$$a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|_2^2$$



$$a(\mathbf{q}, \mathbf{k}) = -\|\mathbf{q} - \mathbf{k}\|$$

Self-Attention

- 만약 queries, keys, values가 모두 같은 값이면 어떨까?
- Token들의 집합이 모두 같은 attention module을 self-attention이라고 부른다.
- Self-Attention은 자기 자신 중 중요한 부분을 attention하는 과정이다.

$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{x}_i$$

입력 $\{\mathbf{x}_i\}$ 자체를 keys와 values로 사용한다

Self-Attention + Multi-Head Attention

- 여러 관점(서로 다른 파라미터)에서 Self-Attention을 수행하면 더 좋을 것이다.
- 이와 같은 multiple Self-Attention은 Multi-Head Attention으로 구현한다.

$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^n) = \sum_{i=1}^n \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

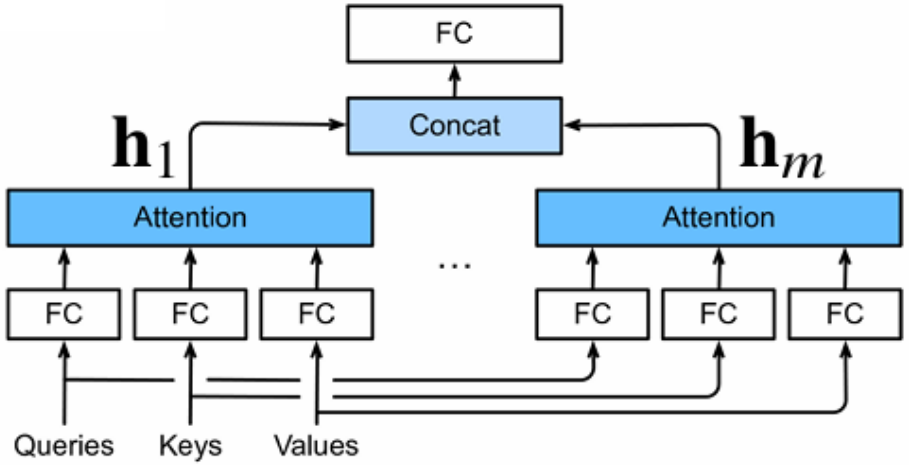
$$\mathbf{h}_m = f(\mathbf{W}_m^{(q)}\mathbf{x}, \{\mathbf{W}_m^{(k)}\mathbf{x}_i, \mathbf{W}_m^{(v)}\mathbf{x}_i\}_{i=1}^n)$$

$$= \sum_{i=1}^n \alpha(\mathbf{W}_m^{(q)}\mathbf{x}, \mathbf{W}_m^{(k)}\mathbf{x}_i)\mathbf{W}_m^{(v)}\mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{x}_i$$

Multiple Self-Attention은

head-wise neural net으로 구현한다



Self-Attention + Multi-Head Attention

- 여러 관점(서로 다른 파라미터)에서 Self-Attention을 수행하면 더 좋을 것이다.
- 이와 같은 multiple Self-Attention은 Multi-Head Attention으로 구현한다.

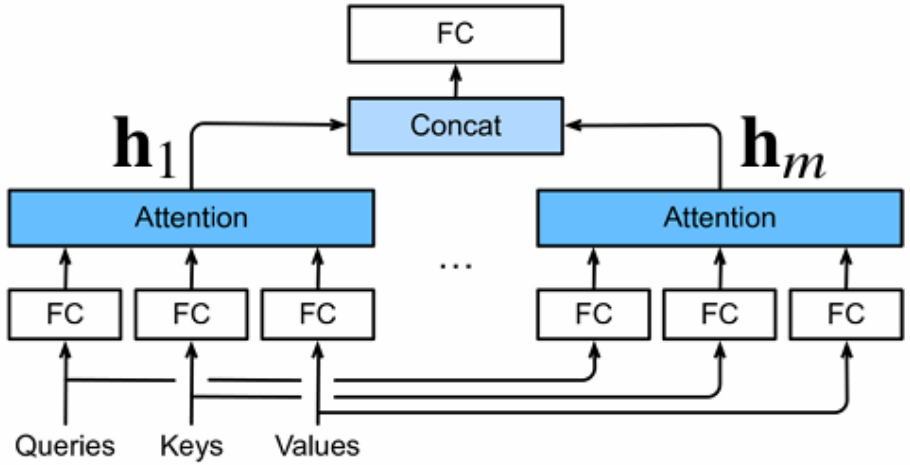
$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

$$\mathbf{h}_m = f(\mathbf{W}_m^{(q)}\mathbf{x}, \{\mathbf{W}_m^{(k)}\mathbf{x}_i, \mathbf{W}_m^{(v)}\mathbf{x}_i\}_{i=1}^{n})$$

$$= \sum_{i=1}^{n} \alpha(\mathbf{W}_m^{(q)}\mathbf{x}, \mathbf{W}_m^{(k)}\mathbf{x}_i)\mathbf{W}_m^{(v)}\mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{x}_i$$

이 파라미터들은 같은 head에 공유된다(shared)



Self-Attention + Multi-Head Attention

- 여러 관점(서로 다른 파라미터)에서 Self-Attention을 수행하면 더 좋을 것이다.
- 이와 같은 multiple Self-Attention은 Multi-Head Attention으로 구현한다.

$$f(\mathbf{x}, \{(\mathbf{x}_i, \mathbf{x}_i)\}_{i=1}^{n}) = \sum_{i=1}^{n} \alpha(\mathbf{x}, \mathbf{x}_i)\mathbf{x}_i$$

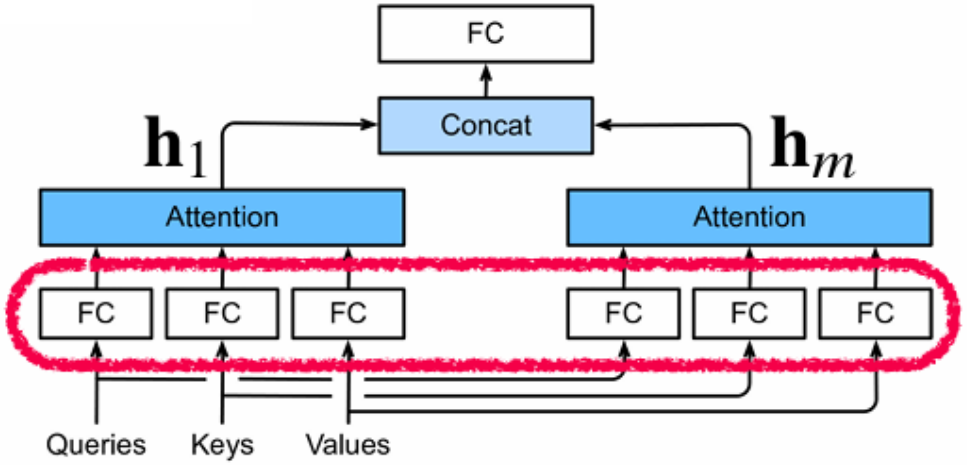
$$\mathbf{h}_m = f(\mathbf{W}_m^{(q)}\mathbf{x}, \{\mathbf{W}_m^{(k)}\mathbf{x}_i, \mathbf{W}_m^{(v)}\mathbf{x}_i\}_{i=1}^{n})$$

$$= \sum_{i=1}^{n} \alpha(\mathbf{W}_m^{(q)}\mathbf{x}, \mathbf{W}_m^{(k)}\mathbf{x}_i)\mathbf{W}_m^{(v)}\mathbf{x}_i$$

$$\mathbf{k}_i = \mathbf{x}_i \quad \mathbf{v}_i = \mathbf{x}_i$$

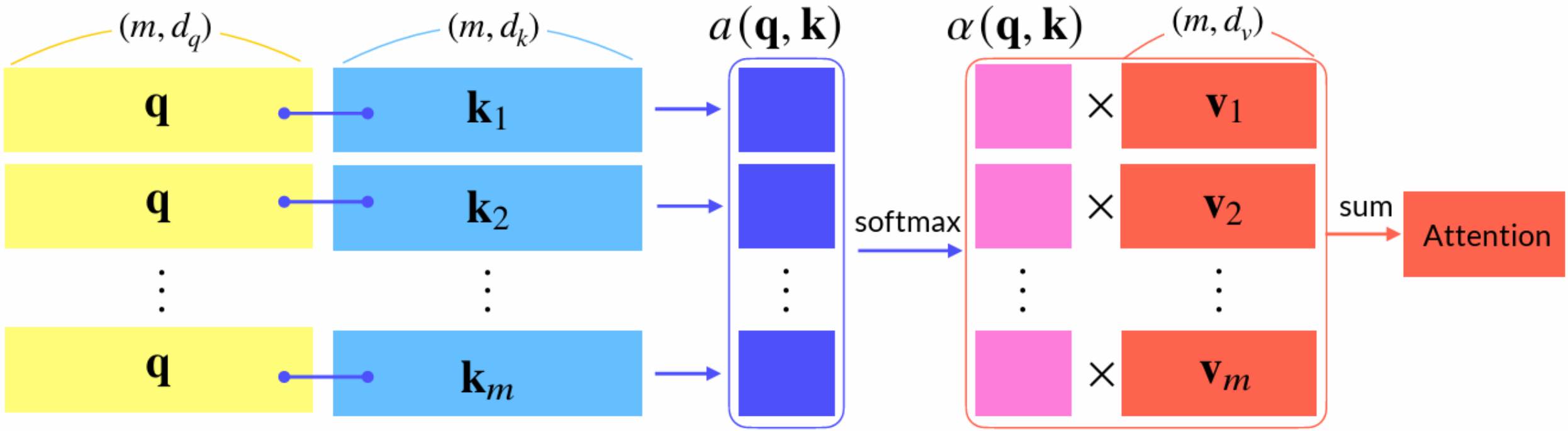
Self-Attention with MHA는

parallel computation에 특화되어 있다



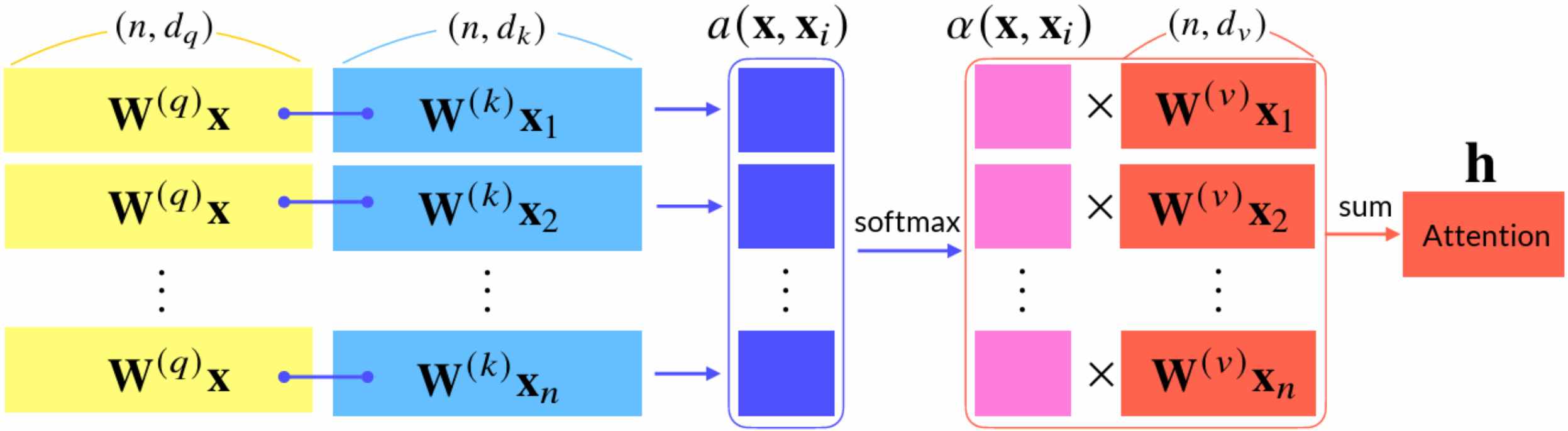
(Recap) Attention Module

$$f(\mathbf{q}, \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^{m}) = \sum_{i=1}^{m} \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{j=1}^{m} \exp(a(\mathbf{q}, \mathbf{k}_j))} \mathbf{v}_i$$

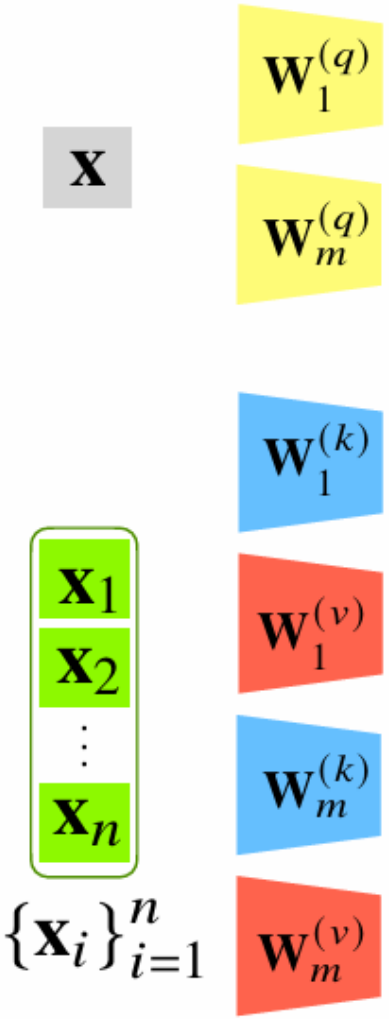


Parallelism in Multi-Head Self-Attention

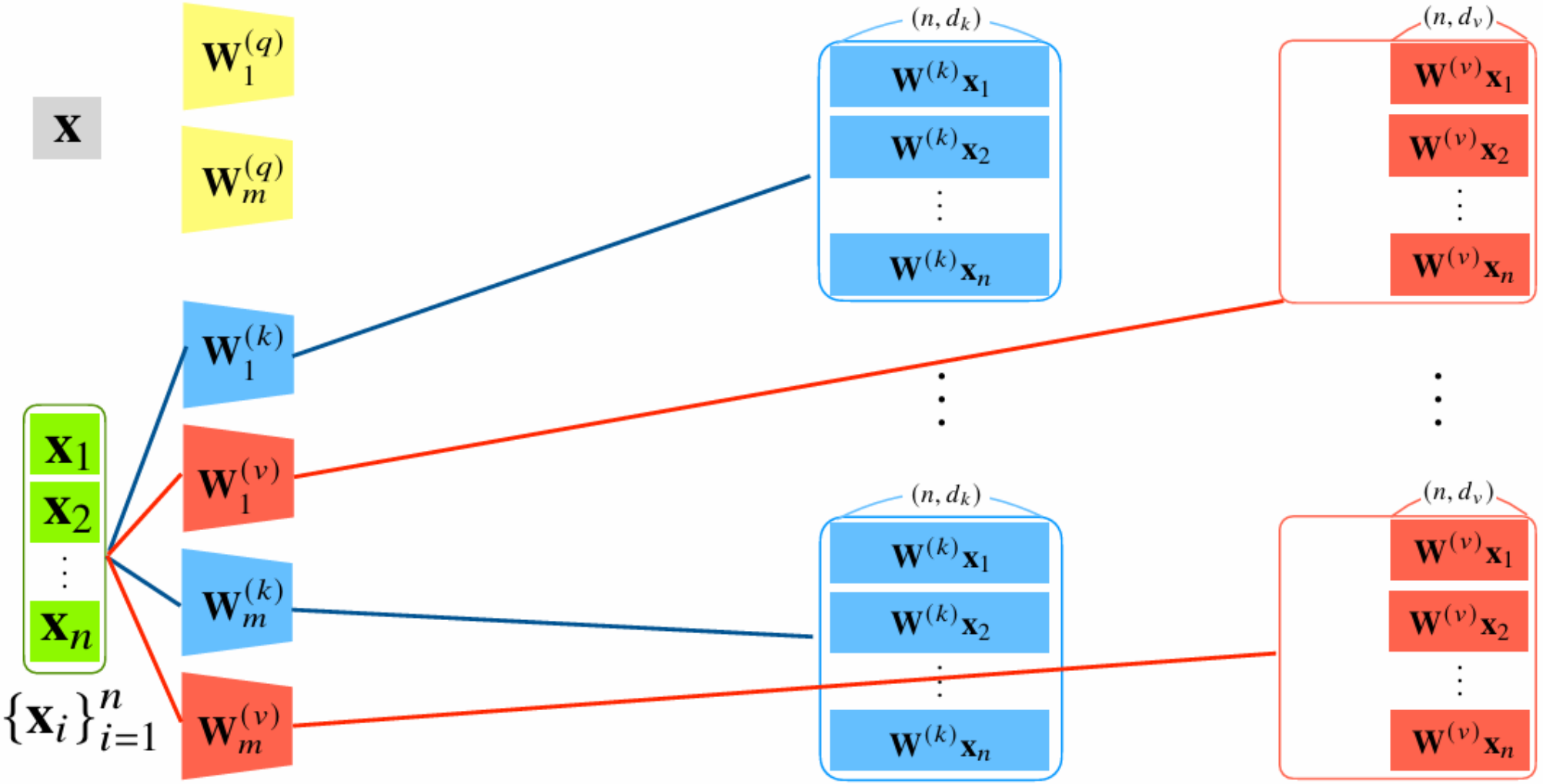
$$f(\mathbf{W}_m^{(q)}\mathbf{x}, \{\mathbf{W}_m^{(k)}\mathbf{x}_i, \mathbf{W}_m^{(v)}\mathbf{x}_i\}_{i=1}^n) = \sum_{i=1}^{n} \alpha(\mathbf{W}_m^{(q)}\mathbf{x}, \mathbf{W}_m^{(k)}\mathbf{x}_i)\mathbf{W}_m^{(v)}\mathbf{x}_i$$



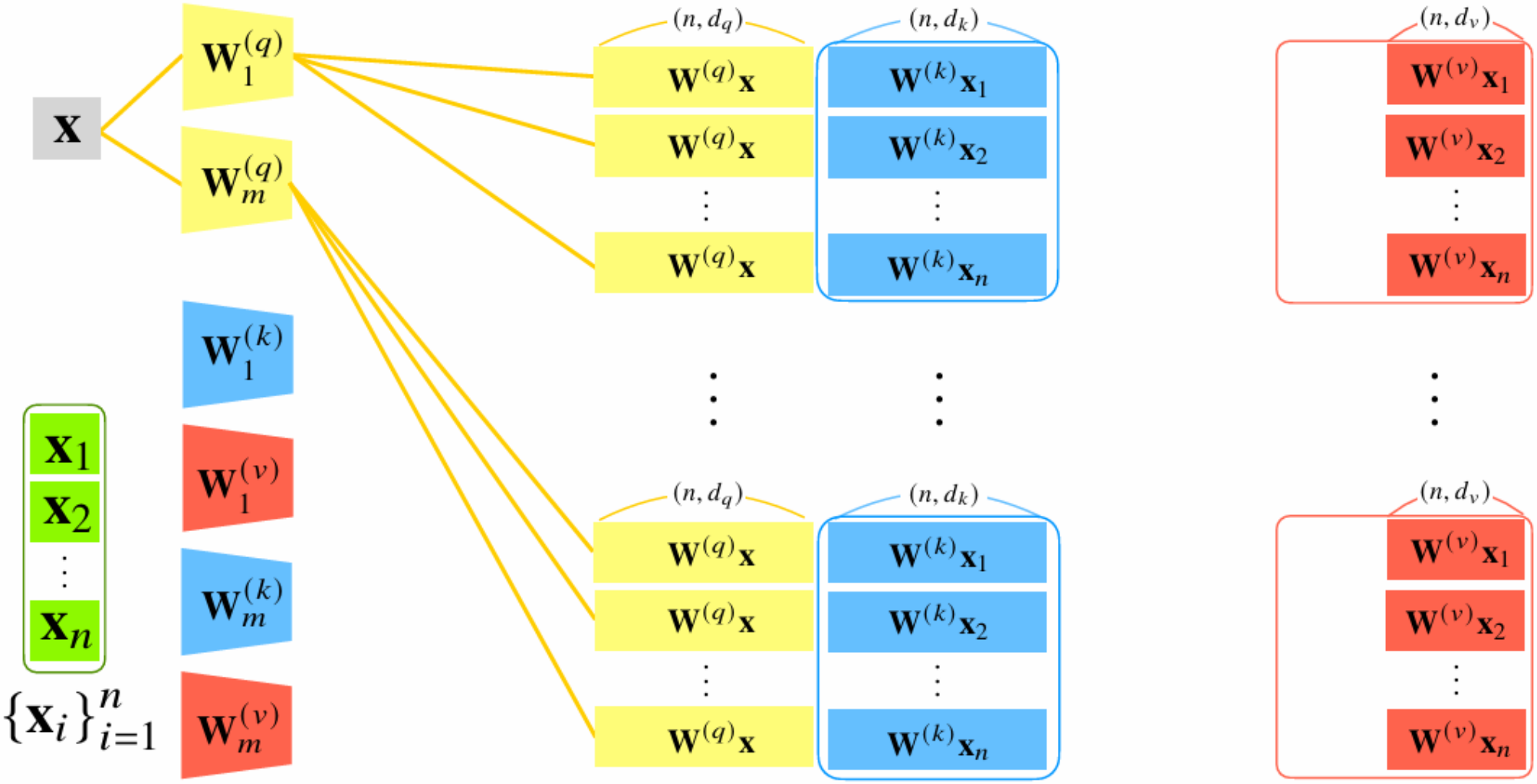
Parallelism in Multi-Head Self-Attention



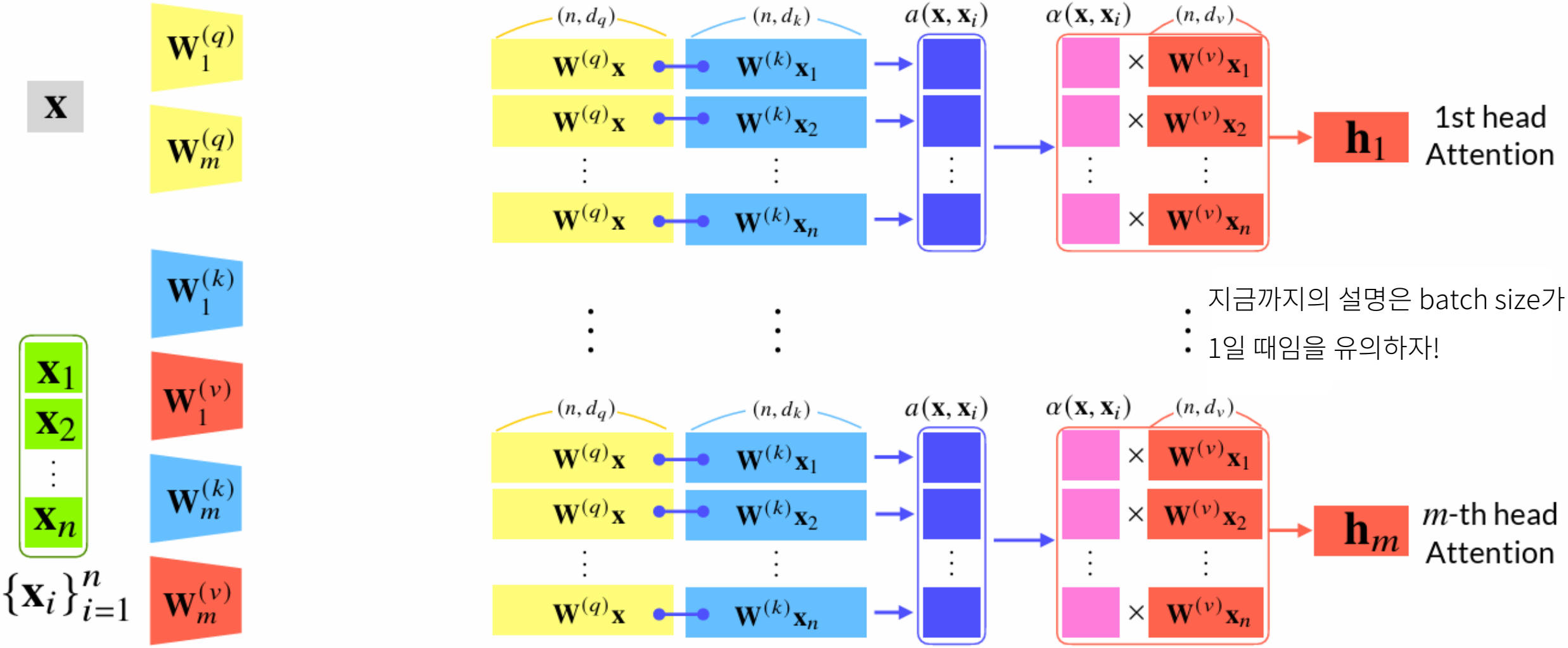
Parallelism in Multi-Head Self-Attention



Parallelism in Multi-Head Self-Attention



Parallelism in Multi-Head Self-Attention



지금까지의 설명은 batch size가 1일 때임을 유의하자!

Problems of Self-Attention

- Self-Attention에는 수학적으로 치명적인 문제가 있는데, 바로 순서를 부여할 수 없다는 점이다.
- 예를 들어 input이 이미지라면, 각 patch가 원래 이미지의 어느 곳에 위치하는 지 self-attention은 알 수 없는 것이다.
- Input이 텍스트라면, “She likes it, but She doesn’t”에서 She과 She을 구분할 수 없는 것이다.
- 이와 같은 속성을 permutation invariant라고 한다.

multi-head self-attention

$$f(\mathbf{z}^{(q)}, \mathbf{z}_i^{(k)}, \mathbf{z}_i^{(v)}\}_{i=1}^n) = \sum_{i=1}^n \alpha(\mathbf{z}^{(q)}, \mathbf{z}_i^{(k)})\mathbf{z}_i^{(v)}$$

permutation

equivalent!

$$f(\mathbf{z}^{(q)}, \mathbf{z}_{\pi(i)}^{(k)}, \mathbf{z}_{\pi(i)}^{(v)}\}_{i=1}^n) = \sum_{i=1}^n \alpha(\mathbf{z}^{(q)}, \mathbf{z}_i^{(k)})\mathbf{z}_i^{(v)}$$

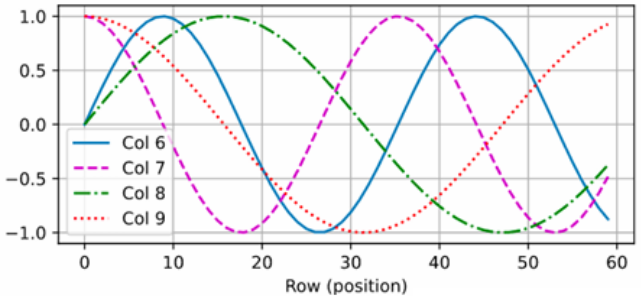
$$\mathbf{z}^{(q)} = \mathbf{W}^{(q)}\mathbf{x}$$
$$\{\mathbf{z}_i^{(k)} = \mathbf{W}^{(k)}\mathbf{x}_i\}_{i=1}^n$$
$$\{\mathbf{z}_i^{(v)} = \mathbf{W}^{(v)}\mathbf{x}_i\}_{i=1}^n$$

Positional Encoding

- Self-Attention의 permutation invariant를 해결하려면 각 토큰에 위치 정보를 부여해주면 된다.
- 이러한 방법을 Positional Encoding (PE)이라고 한다.
- 아래와 같은 Sinusoidal PE는 퓨리에 변환을 기반으로 다음과 같은 위치 정보를 생성하여 각 토큰에 더해준다.

$$f(\mathbf{W}^{(q)}\mathbf{PE}(\mathbf{x}), \{\mathbf{W}^{(k)}\mathbf{PE}(\mathbf{x}_i), \mathbf{W}^{(v)}\mathbf{PE}(\mathbf{x}_i)\}_{i=1}^{n})$$

$$\underset{\text{positional encoding}}{\mathbf{x}_j \mapsto \mathbf{PE}(\mathbf{x}_j)} = \begin{cases} x_j + \sin\left(\frac{i}{10000^{j/d}}\right) & j : \text{even} \\ x_j + \cos\left(\frac{i}{10000^{(j-1)/d}}\right) & j : \text{odd} \end{cases}$$



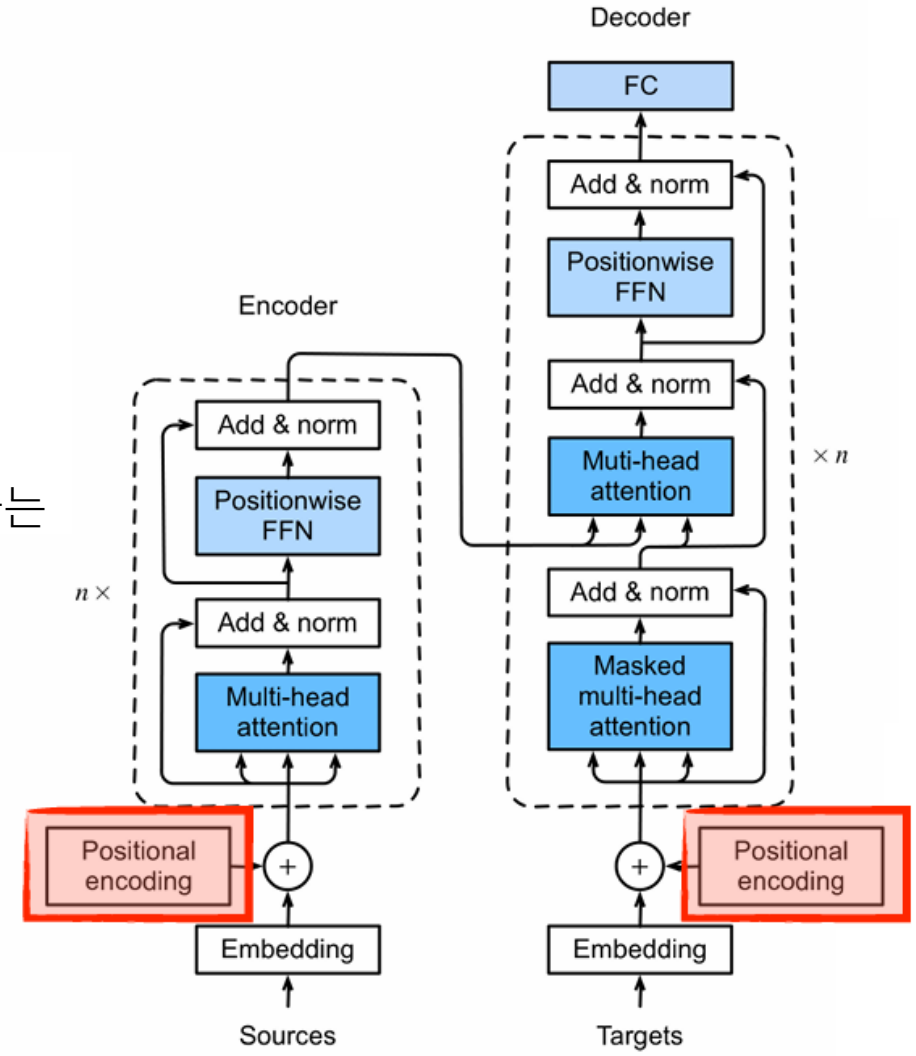
가까운 위치정보끼리는 값이 비슷해야하지만 서로 값이 같아서는 안됩니다.

Positional Encoding에는 다양한 방법이 존재하며 굉장히 활발하게 연구되고 있는 분야입니다.

Attention Is All You Need

Transformer

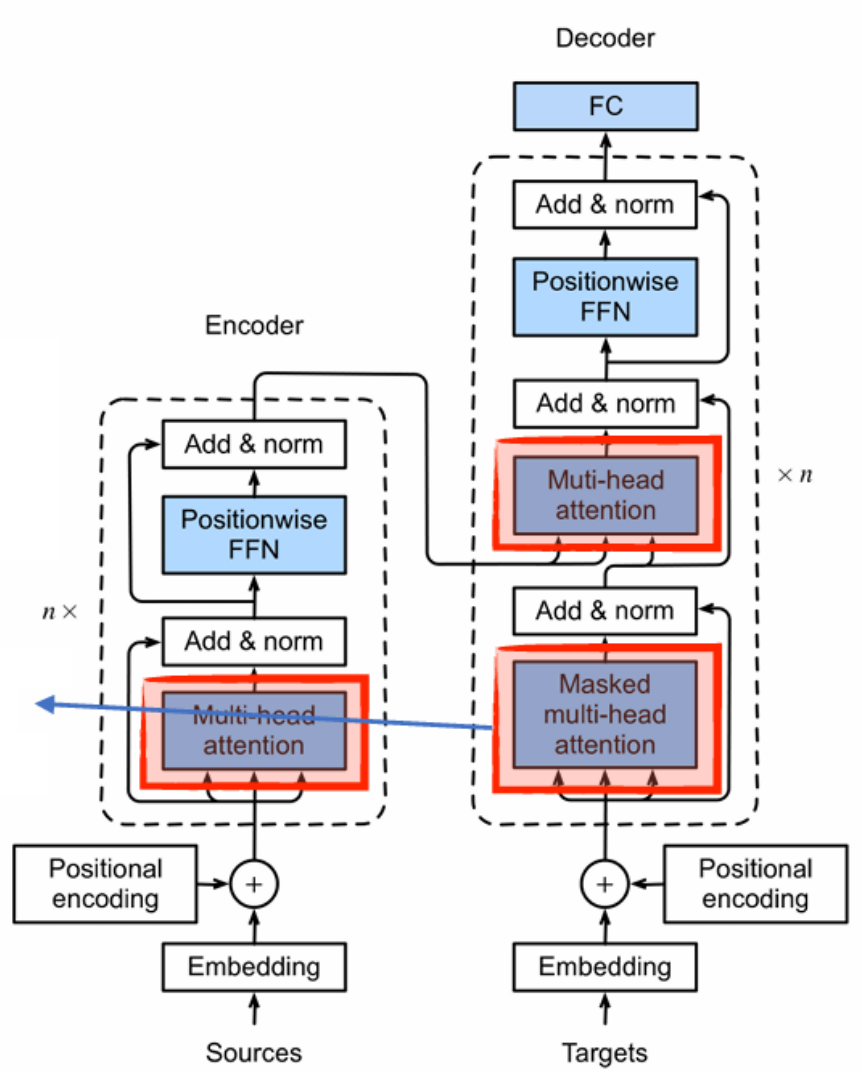
- 2017년 Vaswani 등이 NeurIPS에 Transformer를 발표하였다.
- Transformer는 attention modules을 사용한다.
- 최초의 Transformer는 기계 번역(e.g. 한영 번역)을 위해 고안되었다.
- 기계 번역을 위해서는 한글을 입력받아 정보를 압축하는 Encoder와, 압축된 한글 정보로부터 입력받은 영어 중 가장 확률이 높은 단어를 선택하는 Decoder로 모델을 구성해야 한다.
- Transformer는 총 네 개의 컴포넌트로 구성되어 있다.
  - Positional Encoding (PE)
  - Multi-Head Attention (MHA)
  - Residual connection + Layer Norm
  - Positionwise FeedForward Network (FFN)



Transformer

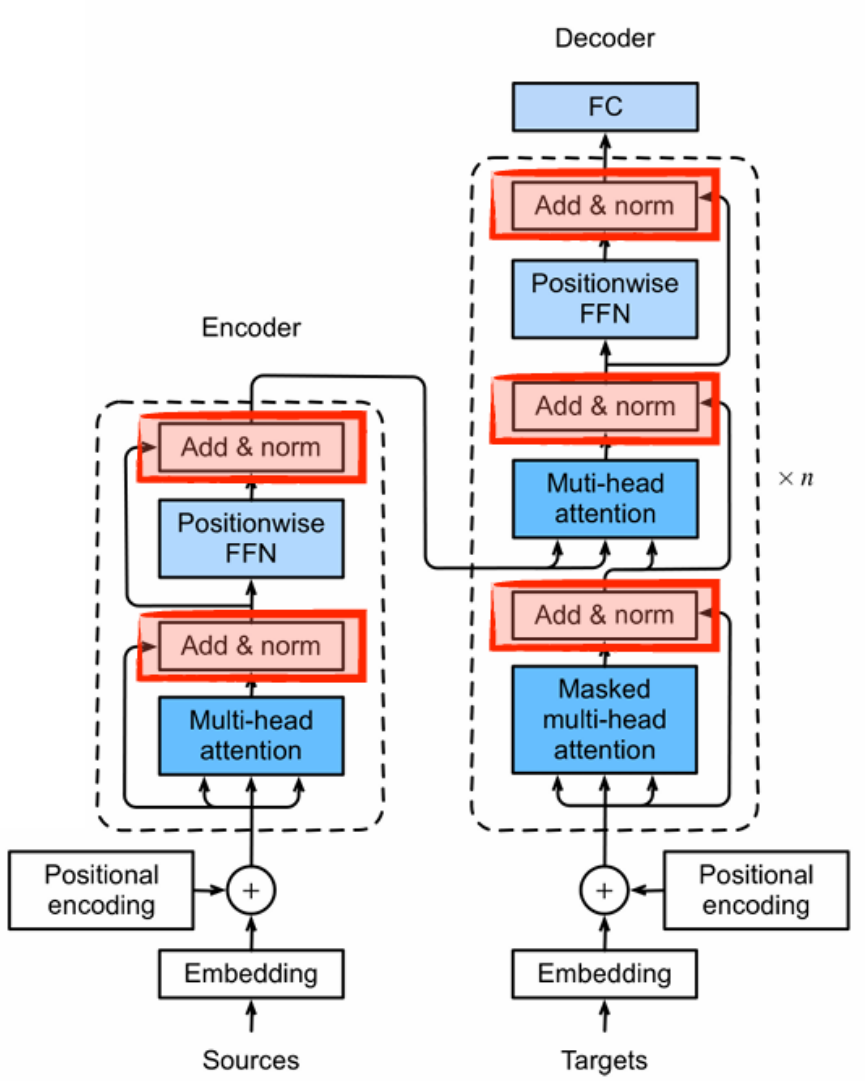
- Transformer는 총 네 개의 컴포넌트로 구성되어 있다.
  - Positional Encoding (PE)
  - Multi-Head Attention (MHA)
  - Residual connection + Layer Norm
  - Positionwise FeedForward Network (FFN)

모델이 다음 예측 정답을 미리 보면 (cheating) 안되므로
현재 이전 토큰만 살려 두고 나머지 토큰은 masking한다



Transformer

- Transformer는 총 네 개의 컴포넌트로 구성되어 있다.
  - Positional Encoding (PE)
  - Multi-Head Attention (MHA)
  - Residual connection + Layer Norm
  - Positionwise FeedForward Network (FFN)



Which scoring function? $< \mathrm{q}, \mathrm{k} >$ vs $||\mathrm{q} - \mathrm{k}||^2$

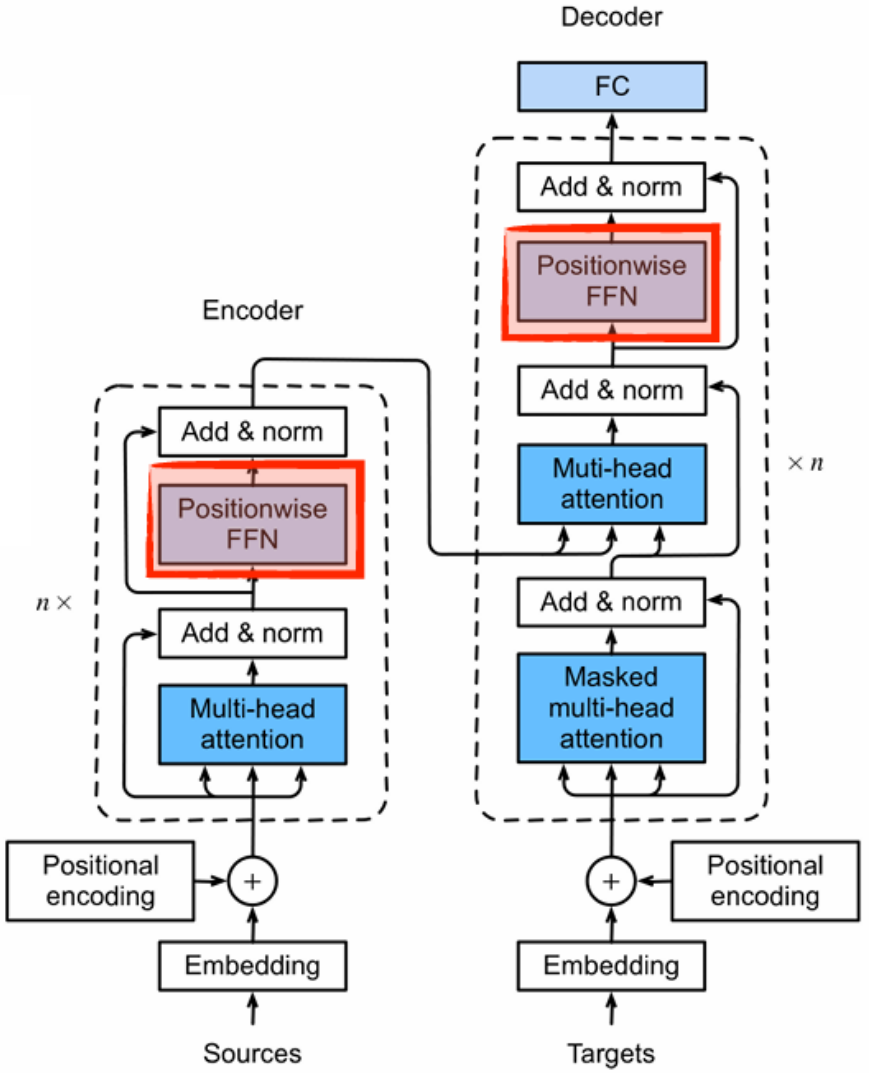
- 만약 key 벡터를 LayerNorm으로 normalize하면 dot-product $< \mathrm{q}, \mathrm{k} >$ 와 gaussian $-||\mathrm{q} - \mathrm{k}||^2$의 output은 동일하다.
- 따라서 LN을 쓴다면 gaussian 대신 dot-product를 사용해 연산량을 줄일 수 있다.

$$\sum_{i=1}^{m} \frac{\exp\left(-\frac{1}{2}||q - k_i||_2^2\right)}{\sum_{j=1}^{m} \exp\left(-\frac{1}{2}||q - k_j||_2^2\right)} = \sum_{i=1}^{m} \frac{\exp\left(-\frac{1}{2}||q||_2^2 + < q, k_i > -\frac{1}{2}||k_i||_2^2\right)}{\sum_{j=1}^{m} \exp\left(-\frac{1}{2}||q||_2^2 + < q, k_j > -\frac{1}{2}||k_j||_2^2\right)}$$

$$= \sum_{i=1}^{m} \frac{\exp\left(-\frac{1}{2}||q||_2^2\right) \exp\left(< q, k_i >\right) \exp\left(-\frac{1}{2}||k_i||_2^2\right)}{\sum_{j=1}^{m} \exp\left(-\frac{1}{2}||q||_2^2\right) \exp\left(< q, k_j >\right) \exp\left(-\frac{1}{2}||k_j||_2^2\right)}$$

Transformer

- Transformer는 총 네 개의 컴포넌트로 구성되어 있다.
  - Positional Encoding (PE)
  - Multi-Head Attention (MHA)
  - Residual connection + Layer Norm
  - Positionwise FeedForward Network (FFN)



실습:

세 가지 Attention module과 Transformer를 구현해보자!

KUGODS

고려대학교
KOREA UNIVERSITY